Technical Report
97-05

Reactive Control of Distributed Interactive Simulations

W. KOHN, J. B. REMMEL AND A. NERODE

August 1997

REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE **31 March 1998**	3. REPORT TYPE AND DATES COVERED **Technical Report**

4. TITLE AND SUBTITLE
Reactive Control of Distributed Interactive Simulations

5. FUNDING NUMBERS
DAAH04-96-1-0341

6. AUTHOR(S)
W. Kohn, J.B. Remmel and A. Nerode

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Regents of the University of California
c/o Sponsored Projects Office
336 Sproul Hall
Berkeley, CA 94720-5940

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U. S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
ARO 35873.78-MA-MUR

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by the documentation.

12 a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12 b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)
In this paper we propose a distributed control architecture for Distributed Interactive Simulation (DIS) applications which allows for the dynamic integration and communication of interacting processes using inteligent control agents cooperating via a logic communication network. The type of DIS applications which we discuss include the interactive simulation initiative of the US Army to create a realistic warfare training environment by using high speed communications to permit real-time interaction between geographically dispersed warfare simulations with actual combat units in the field, cooperative game playing and virtual manufacturing.

19980519 185

14. SUBJECT TERMS
interactive simulation, hybrid systems, control programs, logical and evolutionary specifications

15. NUMBER OF PAGES
59

16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION ON THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev.2-89)
Prescribed by ANSI Std. 239-18
298-102

Technical Report
97-05

Reactive Control of Distributed Interactive Simulations

W. Kohn, J. B. Remmel and A. Nerode

August 1997

Reactive Control of Distributed Interactive Simulations

Wolf Kohn[*]and **Jeffrey B. Remmel**[†]
HyBrithms Corp. [‡]
11201 SE 8th Street #J140
Bellvue, WA 98004-6420
Anil Nerode[§]
Center for Foundations of Intelligent Systems
625 Rhodes Hall, Cornell University
Ithaca, NY 14853-3801

Abstract

In this paper we propose a distributed control architecture for Distributed Interactive Simulation (DIS) applications which allows for the dynamic integration and communication of interacting processes using intelligent control agents cooperating via a logic communication network. The type of DIS applications which we discuss include the interactive simulation initiative of the US Army to create a realistic warfare training environment by using high speed communications to permit real-time interaction between geographically dispersed warfare simulations with actual combat units in the field, cooperative game playing, and virtual manufacturing. Our architecture, termed Multiple Agent Hybrid Control Architecture (MAHCA), is based on recent advancements in hybrid systems theory and applications. Our approach provides for the flexible interoperability of distributed, real-time information systems by generating, in real time, control programs which comply with logical and evolutionary specifications.

1 Introduction

A network of distributed interactive simulations is a hybrid system, that is, a system described by an amalgamation of logical and evolution models. The type of applications that motivated our work here is a distributed interactive simulation initiative of the US Army to create a realistic warfare training environment by using high speed communications to permit real-time interaction between geographically dispersed warfare simulations, with a wide range of capabilities and scale of operation, with actual combat units in the field, see [68,69,70]. Other applications include cooperative game

[*]Research supported by Dept. of Commerce Agreement 70-NANB5H1164. E-mail: `wk@hybrithms.com`

[†]Research supported by Dept. of Commerce Agreement 70-NANB5H1164. E-mail: `jremmel@ucsd.edu`

[‡]Formerly known as Sagent Corporation.

[§]Research supported by Dept. of Commerce Agreement 70-NANB5H1164, and the ARO under the MURI program "Integrated Approach to Intelligent Systems", grant no. DAA H04-96-1-0341. E-mail: `nerode@hybrid.cornell.edu` playing over the Internet, virtual manufacturing and virtual enterprises. We will refer to all these types of applications by the generic name of Distributed Interactive Simulations (DIS).

The importance of realistic training was emphasized by General George C. Marshall to Congress before World War II [70]. Congress supported a sequence of large-scale maneuvers then, starting with the Louisiana Maneuvers in the spring of 1940. More recently, the Army created a new Louisiana Maneuvers (LAM) initiative to make decisions about future doctrine, force mix, force composition, and other fundamental issues. The last twenty years' efforts resulted in many successful simulations for various levels of command and different granularities of precision in modeling battlefield dynamics. Simulation of multiple levels of battlefield dynamics is accomplished by the Corps Battle Simulation (CBS) system for training at the corps/division level and by the Brigade/Battalion Battle Simulation (BBS) system for the brigade/battalion level. However, the National Simulation Center evaluation of their software designs and hardware architectures is that they limit realism. The Warfighter's Simulation 2000 (WARSIM 2000) was established to improve the effectiveness of training by dramatically increasing the realism and scope of the available training environment. The intent is to evolve a system which will synchronize and resolve the interactions of many different entities: e.g., a unit, a weapons system, a minefield, an operation's phase line, or other objects. WARSIM 2000 will use a DIS Network to interface with other simulated and actual units such as the Standard Theater Army Command and Control System (STACCS), Battle Command Training Program (BCTP), Combined Arms Tactical Trainer (CATT), and Battlefield Distributed Simulation-Developmental (BDS-D). WARSIM 2000 is anticipated to be operational in the late 1990s.

In this paper we will discuss a proposed distributed control architecture for a DIS architecture which allows for the dynamic integration and communication of interacting processes using intelligent control agents cooperating via a logic communication network. This architecture, termed the Multiple Agent Hybrid Control Architecture (MAHCA) is based on our recent advancements in hybrid systems theory and applications. This approach provides for the flexible interoperability of distributed, real-time information systems by generating, in real time, control programs which comply with logical and evolutionary specifications.

1.1 Technical barriers to DIS

The type of Distributed Interactive Simulation network we envision is a large-scale, dynamically scalable aggregation of sets of geographically-distributed, highly diverse system simulations interacting over a wide-area, high-speed communication channel. There are a number of the technical issues that must be addressed before such a system can be implemented at an acceptable cost. For example, for the Army DIS system, the system must be dynamically scalable. That is, we must allow individual simulations to be connected and disconnected during a single virtual session. The system must maintain a consistent, accurate view of the synthetic process between simulations with dissimilar communications protocols, data representations, scope of actions, decision time scales and levels of abstraction. The system must provide a framework for the integration of new simulation hardware and software including system verification. Of course such issues will will also be important, to a greater or lesser degree, in other DIS applications as well. Among some of the issues which will come up in all DIS are realism, synchronization, verification, and scalability.

Realism: In a DIS environment, one strives to achieve realism relative to every user, meaning that a user's expected perception of reality and what is delivered to the user by the the network are in agreement modulo some accepted relaxation. We do not propose to deliver global realism because except for trivial scenarios that is not possible. For example, in the Army DIS system, the goals are the evaluation of operational alternatives and training commanders, staffs, and weapons crews on the concept of operations and the operational environment. Synchronization of operations in training and in war depends on a common view of the local battlefield environment appropriate to the level of command (i.e. consistent local views from weapons systems level through higher-level command). One of the known technical shortfalls of the current Army DIS architecture in creating and sharing such a set of views is the inability to correlate environments (e.g. terrain, vegetation, ocean bottom, weather, clouds, communications) at multiple levels of aggregation. For example, we must avoid problems such as a vehicle being "driven" from Fort Knox which goes behind a hill at the National Training Center but does not lose line-of-sight communications or a vehicle generated by a Semi-Automated Forces program which drives smoothly over a 5-feet-deep, 30-feet-wide crater "created" by a cratering charge placed by another program [66]. This lack of correlation between elements in the battle theater decreases realism in representing the effects of terrain and communications on military operations. The current approach to a solution of the problem is to identify critical parameters, coordinate agreement on a distributed communications mechanism, and implement simulations in a common synthetic environment. This approach supports the creation of experiments and the conduct of tests only for a single solution based on current technology to integrate constructive and virtual simulation with live simulations. However, this traditional engineering approach to integration of logical and evolution models will not support cost-effective solutions to integration of diverse methods for achieving spatial and temporal consistency for Advanced Distributed Simulations (ADS) as the technology evolves over time.

Synchronization: A basic need for expansion of the training capabilities present in the existing DIS architecture is the ability to play "what-if drills" with coalition operation alternatives, including peacemaking, peacekeeping, and a range of operations other than war. The synchronization of operations that will make this possible will require substantial horizontal technology integration. Synchronization of operations depends on what is now, according to General Gordon R. Sullivan, the most difficult thing to do: "get a common picture, a common view of the battle" [70].

Verification: One of the most challenging problems facing the successful implementation of DIS systems is the verification of software systems that use both logical (e.g. human behavior representation) and evolution (e.g. environmental representation) models. These models use fundamentally different mathematical tools. Cognitive models are built using linguistic (in the Computer Science sense) tools which depend on models built using the tools of discrete mathematics, e.g. grammars, difference equations, logic, etc. Models of the physical environment are built using simulation tools which support experiments with compositions of both discrete mathematical based, linguistic (logical) models and continuous models involving differential equations, normed vector spaces, etc. This dichotomy is characteristic of hybrid systems. Determining the behavior of the composition of models for safety, reliability and performance constraints requires experimentation. The general approach currently used for verification is to explore design failure modes and track correction of bugs in the software until reaching a comfort level and declaring success. This leaves a nagging expectation that not all of the states of the computer finite-state machine have been visited and tested. And, when new capabilities are added, new failure modes are created so the whole testing process must begin anew. The designs of the existing CBS and BBS simulations makes software modification and validation of software interoperability a lengthy, resource-intensive, medium-risk process. Current verification strategies work well only where the systems are small and extensive testing is feasible. For systems the size and complexity of the Army DIS, exhaustive testing to verify performance for every state of the system finite-state machine is not possible. That is, it is simply not feasible to execute every path under every possible failure. In addition, every modification creates the possibility of new failure modes and thus requires new testing.

Scalability: A DIS architecture must provide a means of explicitly representing multiple time scales of action and for rapid reconfiguration of the representation of environment, resources, and strategies to enable experimentation with alternative scenarios. To provide the necessary levels of realism, the DIS must further provide a consistent visualization across multiple levels of abstraction, and provide reliable, timely interaction between existing and future simulations. Current 'state-of-the-art' design does not provide for the ready integration of simulations which must comply simultaneously with both logical and evolution constraints.

Hybrid systems theory provides a number of tools to deal with such problems. A good overview of the work in hybrid systems can be found in the Springer-Verlag volumes on hybrid systems that has appeared in Lecture Notes in Computer Science series [18],[2], [1], and [3]. Similar problems have also occured in applications of hybrid systems to automated highway systems [10] and to air traffic control [62] and [49]. In the next section, we shall outline a specific hybrid systems architecture for DIS applications.

1.2 A multiple-agent distributed architecture for the DIS

To meet the demands of future DIS applications, we assert that a fundamental improvement in control of distributed interactive processes is required. In the following sections we provide a multiple-agent distributed control architecture for the dynamic integration and communication of DIS processes through a communication network. This architecture is based on the mathematical foundations of hybrid systems theory, developed by the authors [27, 15, 32, 36, 39, 41, 42, 46, 47].

It is not possible to apply hybrid systems theory to existing simulations since rewriting these systems would be an enormous task. Instead, we propose to apply the hybrid systems theory to the aggregation of existing simulations within an architectural framework supporting the dynamic reconfiguration of the system, in response to changes of the synthetic process, by the addition or deletion of real or simulation agents and the reallocation of communication resources.

Our approach is to customize our existing reactive, adaptive, real-time control architecture, termed Multiple-Agent Hybrid Control Architecture (MAHCA) [32, 40, 33] to control the interaction between heterogeneous, distributed simulations. The customization consists chiefly of assigning one or more decision makers, or "agents", to each simulation involved in a DIS process, see Figure 1, and to load each agent's knowledge base with logical and equational information about the simulation assigned to it and about primitives for interaction with other agents. Each agent maintains knowledge of the both the state of the simulation, or unit, and of the synthetic process at an appropriate level of abstraction. The agent responds to requests from the simulation, or unit, and provides information to other agents on the net as necessary.

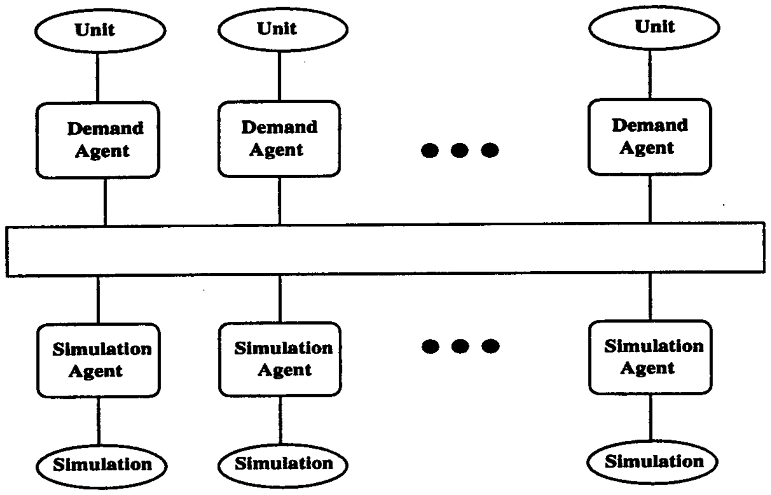


Figure 1: Multiple-Agent Hybrid Control Architecture for DIS

MAHCA implements a dynamic, distributed optimization procedure in which each agent has a cost function and both logic and continuous evolution constraints. Optimum strategies, i.e., feasible plans for all active simulations and users, are inferred by the MAHCA algorithms. Plans and actions are determined simultaneously. The plans so obtained are "decentralized". The MAHCA algorithms also infer required rates of communication for the links in the logic network of agents. MAHCA algorithms are assured to produce plans that are robust and near optimal.

MAHCA is a natural and effective blend of logic programming, the relaxed calculus of variations, infinitesimal differential geometry, and automata theory. Logic Programming is used to express doctrine as a set of facts and rules, and as an inference engine to produce recommended control actions for subunits. The calculus of variations is used for inter-temporal optimization of continuous quantities. The relaxed calculus of variations is the basis for optimization of processes whose evolution has mixed continuous and discrete logical elements, such as costs of deployment and fixed doctrine, since in it one can express constraints of both kinds in identical terms. Infinitesimal differential geometry is the language appropriate for expression of the manifolds on which optimizations take place, for calculation of control actions in the Lie algebra of derivations, and generally for symbolic calculation of optimal plans and budgets. Automata theory is incorporated because near-optimal plans turn out to be calculable using the automata equation calculus of Eilenberg [12].

The underlying mechanism in MAHCA is the computation, on-line, of symbolic solutions for a wide variety of non-convex distributed cost minimization problems. The architecture is a network of distributed agents (subunits); each agent having a local cost function. The agents communicate to other agents only corrections for the agent cost function. Each agent acts at all times to optimize its own cost function using the MAHCA algorithms at its local computer.

Contrary to the older variational calculus and operations research techniques, relaxed plans and control action strategies that are optimal almost always exist on the carrier manifold. But there is a real 'catch'. That is, such optimal plans recommend, based on your current state, not as a specific control action (these are recommended changes in rates of the variables of the simulation), but a probability distribution of control actions instead. But an agent in a state has to take a specific control action, that is, make a non-fuzzy decision. What can this recommended probability distribution on control actions possibly mean? For example, if two control actions "c" and "d" are recommended for the next interval T of time with respective probabilities 1/5, 4/5, and a decision as to control action has to be made, what do you do? The inability to answer how to interpret and implement such recommendations as this held up the implementation in applications of the relaxed variational optimal solutions for at least twenty years. The answer was discovered by Kohn, and subsequently developed by the authors, and is the basis for MAHCA.

What the MAHCA algorithms reveal is that one should relinquish the search for the 'Holy Grail' of implementing a perfectly optimal plan achieving minimal cost, which is what the recommended probability distribution represents. Instead, we implement an approximation to that plan. That is, we lower our expectations and specify what deviation from minimal cost is acceptable for each agent. Then the theorems and symbolic algorithms supporting the MAHCA kernel will compute plans for each agent that make a definite decision (control action) as to which control action to take for the next time period, T, based on what state you are in. If that suboptimal plan is followed, a trajectory evolves for each agent within the desired tolerance of minimality for that agent's cost function, and which meets global goals and constraints. But there is a subtlety, namely, a change of scale is needed, concerning what is an atomic control action. For example, if, according to the optimal plan, the recommendation is the probability distribution on actions described before, then the definite decision action of the nearly optimal plan is to recommend a time interval, T, over which to perform the next control action and to choose the control action over that interval as follows. Divide the interval T in the ratio one to four into two subintervals. Use the control action assigned probability 1/5 for the 1/5 T length subinterval, control action assigned probability 4/5 for the 4/5 T length subinterval (If T is sufficiently small, order does not matter). This is called a "chattering control" in the control literature. The algorithms start with the allowable deviation from optimality and eventually compute the required T and the finite distribution of control actions as above, which guarantee the deviation is not exceeded. The change in scale is that the recommended action for the next interval T is a sequence of short control actions, not a single control action of duration T. This use of refined scales of control is the source of strength of the relaxed variational calculus and of MAHCA. It is based on convex analysis and a generalized form of dynamic programming. This is the central mechanism behind the implementation of real-time computing in MAHCA.

The feature of MAHCA which permits us to implement real-time computing of plans and establish MAHCA advantage over other methods which might be used is based on hybrid dynamic programming [42]. Our recent analysis of the underlying evolution manifold of the system, called carrier manifold [41], shows that as one moves along an optimal trajectory on the carrier manifold in a field of extremals, there is an affine connection associated with the integral curves or flows of the extremal field. The Christoffel symbols for the affine connection can be computed from the Bellman-Hamilton-Jacobi equation of the calculus of variations by comparing tangent planes at infinitely near points on the extremal to get a linear invertible transformation of tangent planes at the two infinitely near points. It is to be noted that these are not connections on the whole manifold in the usual sense. They parallel transport along extremals in an extremal field, but not along arbitrary curves. This can be thought of as an extension of the Caratheodory view of variational calculus as the theory of an extremal field. Our embedding of the optimization problem allows us to apply this optimization techniques to both physical and cognitive models.

The relation obtained is an infinitesimal relation, that is, a single symbolic formula that can be computed in advance using generalized Christoffel symbols. Since we handle control actions entirely by the infinitesimal method as generators of flows or derivations, we can integrate the infinitesimal relation for the connection (a generalization of the geodesic equation of Riemannian geometry) to transport the desired control at the end process when the goal should be met all the way back to the present time to compute a current control, that is, to compute currently needed plans. It is the fact that the infinitesimal relation for the connection can be computed symbolically in advance all at once, that reduces on-line computation when goals change and plans and budgets have to be changed in response. Parallel transport does the trick, in almost completely precomputed symbolic form. We have found only a few special cases of connections in the control literature. Basically, in such cases, the author observes that there is a connection associated with a control found in some other way. Most of these examples are found in mechanics where there are connections, in the usual sense, which allow parallel transport on all curves on a manifold, not just on extremals as ours do.

MAHCA provides a knowledge-based, formal implementation framework for deducing on-line feedback control and reactive strategies for processes involving multiple decision makers (or agents) through the implementation of a constructive algorithm for building automata which simultaneously comply with logical and evolution equations. Each agent of the architecture has the capability for structural adaptation as a function of predictable and unpredictable events and, therefore, can satisfy quality of service and performance requirements in the unavoidable presence of sensory and knowledge uncertainty. Furthermore, if the logic or evolution models are not completely compatible with the system they are modeling, the architecture provides for formal mechanisms for detecting the incompatibility and for tuning the structure of both the logic and evolution models.

Our multiple-agent architecture provides a formal framework for a dynamically reconfigurable distributed information system cooperating without an umpire. Agents can be dynamically spawned or removed from the net. Each agent maintains an appropriate level of knowledge about other agents on the net. In addition, individual agents are self-aware, in that the knowledge base contains information on the hardware capabilities and current status. The capability information from each agent within the architecture is communicated to every other agent. This net wide self awareness provides the capability to adapt to system changes, whether it is the spawning of new agents, introduction of future simulation technology or failures of an existing simulation.

Our multiple-agent control architecture encapsulates each active element on the net. The agents are "trusted" in that they prevent any action by the simulation, or unit, from violating the integrity of the system. To the extent that the high-level (logic) models are "trusted" and low-level (evolution) models are "trusted", we can construct automata which are consistent with the constraints of both models. Testing to determine system equilibrium is still required as is the need to experimentally verify that the high-level logic meets the needs of the users. However, the need for exhaustive testing to experimentally analyze the result of combining high-level logic and low-level evolution representations is not required.

The rest of this paper is organized as follows. In Section 2, we discuss our generic model of a DIS process based on a general conservation criterion. In Section 3, we present the salient features of a

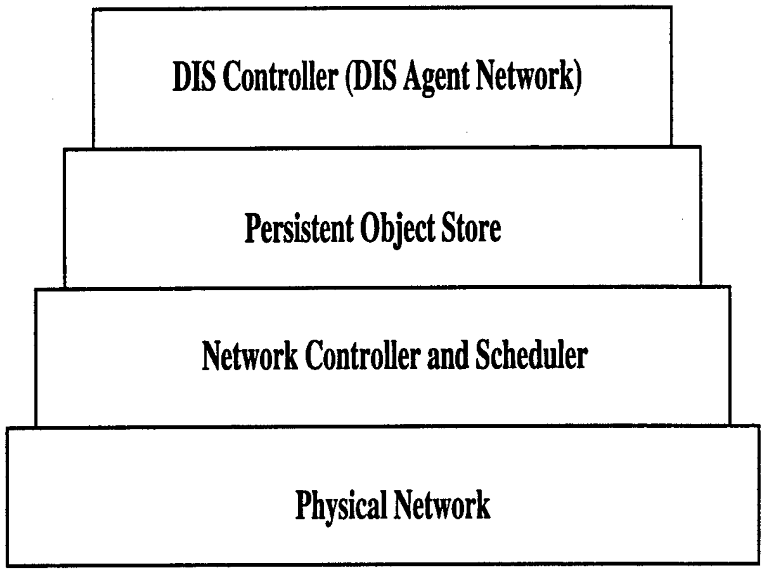

DIS Controller (DIS Agent Network)
Persistent Object Store
Network Controller and Scheduler
Physical Network

Figure 2: Hierarchical Organization of the network

customized version of MAHCA as the "intelligent control" for DIS. In section 4, we will discuss our approach to various synchronization and communications issues which arise in a DIS application. In section 5, we shall describe a simulation of a limited version of the target engagement problem for a dynamic battlefield which incorporates some of the features of our DIS architecture. Finally in section 6, we present some preliminary conclusions.

2 Process Model

Our model for DIS processes builds on the current hierarchical organization of the network operation by adding a set of distributed decision agents for controlling the simulation processes. We shall discuss several aspects of this model in the section, namely the hierarchical organization of the model, the ability of the network of DIS agent to spawn new agents and eliminate existing agents, and the underlying mathematical model of a MAHCA agent.

2.1 Hierarchical organization

The hierarchical organization of the network system is similar to the ISO model, see Figure 2, and conforms to the evolving vision of the DIS architecture [66]. The bottom of the hierarchy contains the Physical Network, composed of the primary and secondary network links, switching nodes, interfaces, etc. The next level is the Network Controller and Scheduler which processes requests from the network for DIS services and queries the data base system for setup and status information. It also receives state information from the network and processes it to generate updates to an aggregated operational network model that resides in the database.

The Persistent Object Store System (POSS) is a dynamic data base providing mechanisms for data store, data flow and presentation at a level of aggregation compatible with the needs of the DIS Controller. This database includes interactive interfaces with the Network Controller and DIS Controllers. The data base also includes entries which comply with the current standards and needs of specific simulations. More generally, data base should also include entries required to support flexible interoperability of reconfigurable simulations. For MAHCA, the objects in the database are simulation generated trajectories that are assembled from generic primitives in accordance with the DIS Controller action commands.

At the top of the hierarchy, the DIS Controller generates control actions to initiate, facilitate, monitor, and adaptively regulate DIS processes, see Figure 1. In order to accommodate the flexible reconfiguration of interoperable simulations, we propose that the network be controlled by a network of autonomous agents, the DIS Agent Network. The DIS Agent network is composed of a (varying) number of devices called ***decision agents*** (agents, for short) implicitly coordinated through a variable (over time) configurable logic communication network implemented through the hierarchy of Figure 2.

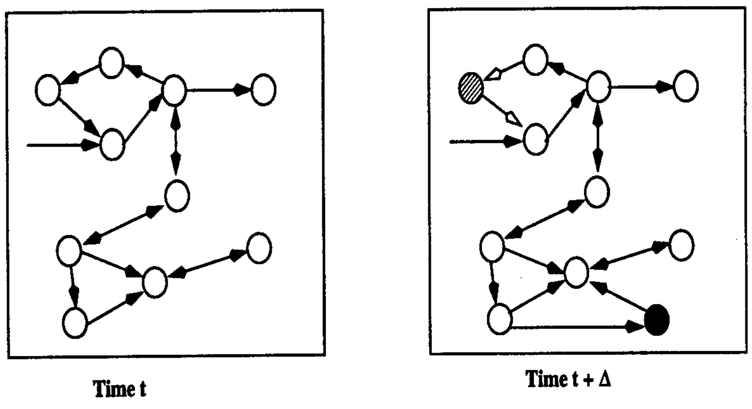
2.2 Dynamic Distribution of Decision Agents

The DIS controller domain of action is a hybrid distributed model of the active DIS processes throughout the network. This model is termed hybrid because its structure is an amalgamation of logic and evolution elements, see [39, 40, 41]. The evolution elements are encoded in the network model stored in the POSS relational data base. The logic elements are encoded in the knowledge base of the individual agents. We propose to model the DIS process dynamics in terms of the conservation of the flow of a commodity called Simulation Service Demand (demand for short) through the logic communication network. This network is composed of connection links and agents. The agents are allocated at the nodes of the network. As a function of time, t, the network configuration varies because new agents are spawned and become part of the network or old ones drop out.

This is illustrated in Figure 3. In Figure 3 circles represent the agents and directed edges represent the logic links between them. We emphasize that the logic connections between the agents are actually mediated through the protocol associated with the hierarchy of Figure 2.

The agents in the logic communication network are of two types: Permanent and Transitory agents. Permanent agents are those associated with DIS processes. They remain active even if the process they control becomes inactive. Transitory agents, which are spawned by the permanent agents, remain part of the network as long as the process in which they participate is active. Transitory agents are spawned as a response, among others, to increased demand, satisfaction of service requirements, and network malfunctions. In Figure 3, edges attached to only one node represent flow of demand in or out of the network.

With MAHCA algorithms, the messages passed between agents in the logic communication network are not raw data. All messages to and from an agent are mainly single corrections for a cost function that the agent or subunit uses for its local optimization. Each agent is given adequate local computing power to solve approximately its cost minimization problem in real-time based on these communicated corrections. The communications bottleneck created by massive raw data transfer



● Spawned Agent ▨ Deactivated Agent ○ Active Agent

—▷ Deactivated Link —▶ Active Link

Figure 3: Logic Communication Network : Transition example

is thus avoided by increasing local computer intelligence for each agent. An agent's computing power required to employ MAHCA is now relatively inexpensive; requiring only a single workstation for most simulations. Distribution of intelligent optimization is our method of addressing the information flow bottleneck for dynamic distributed DIS synchronization and control.

2.3 Model Overview

At each time t, the state of the network is given by a point in a locally smooth manifold M [7]. We refer to M as the *carrier manifold* of the DIS system. The carrier manifold is a dynamic database that captures the hybrid nature of distributed simulation processes. In general, a hybrid system has a hybrid state, the simultaneous dynamical state of all plants and digital control devices. Properly construed, the hybrid states will form a differentiable manifold which we call the carrier manifold of the system. To incorporate the digital states as certain coordinates of points of the carrier manifold, we "continualize" the digital states. That is, we view the digital states as finite, real-valued, piecewise-constant functions of continuous time and then we take smooth approximations to them. This also allows us to consider logical and differential or variational constraints on the same footing, each restricting the points allowed on the carrier manifold. In fact, all logical or discontinuous features can be continualized without practical side-effects. This is physically correct since for any semiconductor chip used in an analog device, the zeros and ones are really just abbreviations for sensor readings of the continuous state of the chip. Every constraint of the system, digital or continuous, is incorporated into the definition of what points are on the carrier manifold. Lagrange constraints are regarded as part of the definition of the manifold as well, being restrictions on what

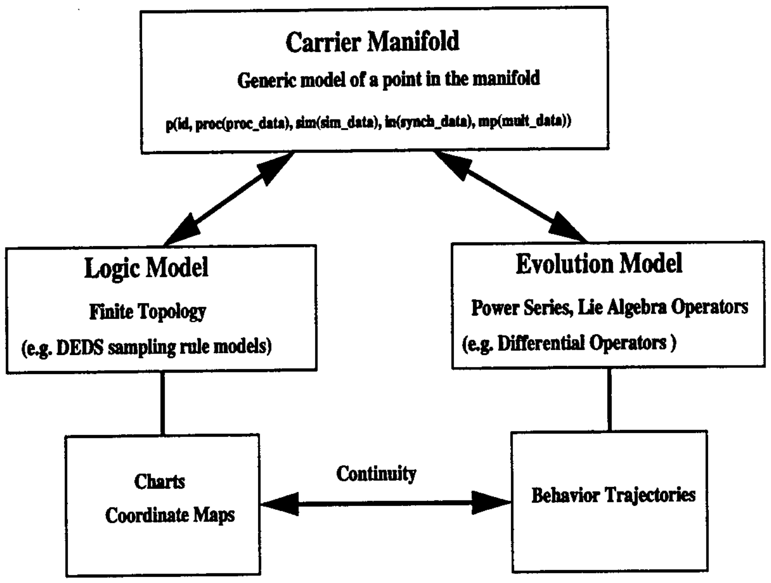

Carrier Manifold
Generic model of a point in the manifold
p(id, proc(proc_data), sim(sim_data), in(synch_data), mp(mult_data))
Logic Model
Finite Topology
(e.g. DEDS sampling rule models)
Evolution Model
Power Series, Lie Algebra Operators
(e.g. Differential Operators)
Charts
Coordinate Maps
Continuity
Behavior Trajectories

Figure 4: Carrier Manifold

points are on the manifold.

The evolution of a distributed simulation process is characterized by two interacting models: an Evolution Model and a Logic Model, see Figure 4. The evolution model generates certain behavior trajectories in M (functions of time taking values in M) referred to as *demand functions.* The logic model encodes the knowledge requirements for the simulation application being executed. This includes the characteristics of the simulations involved, goals of the user(s), synchronization requirements, and communication constraints.

Associated with each point p in the carrier manifold M, there is a finite dimensional vector space called the tangent space at p and termed TM_p. The actions generated by agents are elements of this vector space. The actions, that we will describe in the next subsection, are infinitesimal feedback operators which determine the behavior trajectories in M.

The central constraint on the behavior trajectories is that they satisfy the requirements encoded in the logic model. This is equivalent to the requirement that they be continuous in the finite topology of the carrier manifold, see [32, 40, 38]. This topology is determined by the encoded knowledge. We will discuss this aspect in the next subsection.

2.4 A MAHCA Agent's Model

Let A_i, $i = 1, \dots, N(t)$ denote the agents active at the current time t. In our model, t takes values on the real line. At each time t, the status of each agent in the network is given by a point in a locally differentiable manifold M by a data structure of the form:

$$p(id, proc(proc\_data), media(media\_data), in(synch\_data), mp(mult\_data)) \qquad (1)$$

Here *id* is an identifier taking values in a finite set ID, *proc*() is a relation characterizing the Distributed Interactive Simulation processes status which depends on a list of parameters labeled *proc_data*, which define the load and timing characteristics of the process involved. (Note that the existing simulation processes will normally have different data structures.) Next *media* is a relation that captures attributes of the communication processes involved in the simulation application in operation. It depends on a list of parameters labeled *media_data* whose entries encode constraint instances and protocols of the application at a level of abstraction compatible with the logic communication network. Also in (1), *in*() is a relation carrying synchronization information of the logic communication network with respect to the hierarchical organization of Figure 2. Specifically, it characterizes the protocol at the operation point. This includes information such as priority level, connectivity and time constants. Finally, the relation *mp*() carries multiplicity information, that is, it represents the level of network usability at this point. The associated parameter list, *mult_data*, is composed of statistical parameters reflecting the network's load.

The parameter lists in the data structure of the points of M are composed of integers, such as number of users, reals, such as traffic loads, and discrete values, such as process identifiers or switches. They characterize the status of the network and the active processes. Computing the evolution of these parameters over time is the central task of the model.

The demand function D_i of an active agent A_i is given by a continuous function,

$$D_i : M \times T \rightarrow R^+ \tag{2}$$

where T is the real line (time space) and R^+ is the positive real line.

From an agent's point of view, the dynamics of the control network is characterized by certain trajectories on the manifold M. These trajectories characterize the flow of information through the network and its status. Specifically, we need to define two items:

(i) A generator for the demand functions at time t,

$$\{D_i(p,t) : i \in I(t), p \in M\} \tag{3}$$

where $I(t)$ is the set of active agents at time t and

(ii) the control actions issued by the agents.

We will see shortly that these actions are implemented as infinitesimal transformations defined in M. The general structure of the demand function in (2) for an agent A_i at time t is

$$D_i(p,t) = F_i(U_i, D, \alpha_i)(p,t) \tag{4}$$

where F_i is a smooth function, D is the vector of demand functions, C_i^u is the i-th unsatisfied demand function, and α_i is the command action issued by the i-th agent. We will devote the rest of this subsection to characterizing this model.

We start with a discussion of the main characteristics of the manifold M. In general a manifold M is a topological space (with topology Θ) composed of three items:

(a) A set of points, for example in our case we will consider points of the form of (1).

(b) A countable family of open subsets of M, $\{U_i\}$ such that

$$\bigcup_i U_i = M.$$

(c) A family of smooth homeomorphisms, $\{\phi_i : \phi_i : U_i \to V_i\}$, where for each j, V_j is an open set in R^k. The sets U_i are referred to in the literature as coordinate neighborhoods or charts. For each chart U_j the corresponding function ϕ_j is referred to as its coordinate chart.

The coordinate chart functions satisfy the following additional condition:

Given any charts U_i and U_j such that $U_i \cap U_j \neq \emptyset$, the function $\phi_i \circ \phi_j^{-1} : \phi_j(U_i \cap U_j) \to \phi_i(U_i \cap U_j)$ is smooth.

In the literature, one usually finds an additional property, which is the Hausdorff property in the definition of manifolds [39]. Since this property does not hold in our application we will not discuss it.

Now we proceed to customize the generic definition of the manifold to our application. We start with the topology Θ associated with M. We note that the points of M have a definite structure, see (1). Suppose the number of these parameters equals k. The knowledge about these parameters is incorporated into the model by defining a finite topology Ω on R^k [13].

The open sets in Ω are constructed from the clauses encoding what we know about the parameters. The topology Θ of M is defined in terms of Ω as follows. For each open set W in Ω such that $W \subseteq V_j \subseteq R^k$, we require that the set $\phi_j^{-1}(W)$ be in Θ. The sets constructed in this way form a basis for Θ so that a set $U \subseteq M$ is open if and only if for each $p \in U$, there is j and an open set $W \in \Omega$ such that $W \subseteq V_j$ and $p \in \phi_j^{-1}(W)$.

To characterize the actions commanded by a MAHCA agent we need to introduce the concept of derivations on M. Let F_p be the space of real valued smooth functions f defined in a neighborhood a point p in M. Let f and g be functions in F_p. A *derivation* v of F_p is a map

$$v : F_p \to F_p$$

that satisfies the following two properties:

$$v(f+g)(p) = (v(f) + v(g))(p) \qquad \text{(Linearity)} \qquad (5)$$
$$v(f \cdot g)(p) = (v(f) \cdot g + f \cdot v(g))(p) \qquad \text{(Leibniz Rule)} \qquad (6)$$

Derivations define vector fields on M and a class of associated curves called integral curves [40]. Suppose that C is a smooth curve on M parameterized by $\psi : I \to M$ where I a subinterval of R. In local coordinates, $p = (p^1, ..., p^k)$, C is given by k smooth functions $\psi(t) = (\psi^1(t), \ldots, \psi^k(t))$ whose derivative with respect to t is denoted by $\dot{\psi}(t) = (\dot{\psi}^1(t), \ldots, \dot{\psi}^k(t))$. We introduce an equivalence relation on curves in M as the basis of the definition of tangent vectors at a point in M [25]. Two curves $\psi_1(t)$ and $\psi_2(t)$ passing through a point p are said to be equivalent at p (notation: $\psi_1(t) \sim \psi_2(t)$), if there exists $\tau_1, \tau_2 \in I$ such that

$$\psi_1(\tau_1) = \psi_2(\tau_2) = p \qquad (7)$$
$$\dot{\psi}_1(\tau_1) = \dot{\psi}_2(\tau_2). \qquad (8)$$

Clearly, $\sim$ defines an equivalence relation on the class of curves in M passing through p. Let $[\psi]$ be the equivalence class containing ψ. A *tangent vector* to $[\psi]$ is a derivation, $v|_p$, such that in local coordinates $(p^1, \ldots, p^k)$, it satisfies the condition that given any smooth function $f : M \to R$,

$$v|_p(f)(p) = \sum_{j=0}^{k} \psi^j(t) \frac{\partial f(p)}{\partial p^j} \tag{9}$$

where $p = \psi(t)$. The set of tangent vectors associated with all the equivalence classes at p defines a vector space called the *tangent vector space* at p, denoted by TM_p. The set of tangent spaces associated with points in M can be "glued" together to form a manifold called the *tangent bundle* which is denoted by TM.

$$TM = \bigcup_{p \in m} TM_p$$

For our purposes, it is important to specify explicitly how this gluing is implemented. This will be explained below after we introduce the concept of a vector field and discuss its relevance in the model.

A *vector field* on M is an assignment of a derivation $v|_p$ to each point p of M which varies smoothly from point to point. That is, if $p = (p^1, ..., p^k)$ are local coordinates, then we can always write $v|_p$ in the form

$$v|_p = \sum_j \lambda^j(p) \frac{\partial}{\partial p^j} \tag{10}$$

Then v is a vector field if the coordinate functions λ_i are smooth.

Comparing (9) and (10) we see that if ψ is a parameterized curve in M whose tangent vector at any point coincides with the value of v at a point $p = \psi(t)$, then in the local coordinates $p = (\psi^1(t), \ldots, \psi^k(t))$, we must have

$$\dot{\psi}^j(t) = \lambda^j(p) \text{ for } j = 1, \ldots, k. \tag{11}$$

In our application, each command issued by the MAHCA agent is implemented as a vector field in M. Each agent constructs its command field as a combination of 'primitive' predefined vector fields. Since the chosen topology for M, Θ, is not metrizable, we cannot guarantee a unique solution to (11) in the classical sense for a given initial condition. However, they have solutions in a class of continuous trajectories in M called *relaxed curves* [50]. In this class, the solutions to (11) are unique. We discuss the basic characteristics of relaxed curves as they apply to our process control formulation and implementation in Section 3. Next, we describe some of their properties as they relate to our plant model and control process. For this objective, we need to introduce the concept of flows in M.

If v is a vector field, any parameterized curve passing through a point p in M is called an ***integral curve associated with*** v, if in local coordinates (11) holds. An integral curve associated with a field v, denoted by $\Psi(t, p)$ is termed the flow generated by v if it satisfies the following properties:

$$\begin{aligned} \Psi(t, \Psi(\tau, p)) &= \Psi(t + \tau, p) \quad &\text{(semigroup property)} \\ \Psi(0, p) &= p \quad &\text{(initial condition)} \end{aligned} \tag{12}$$

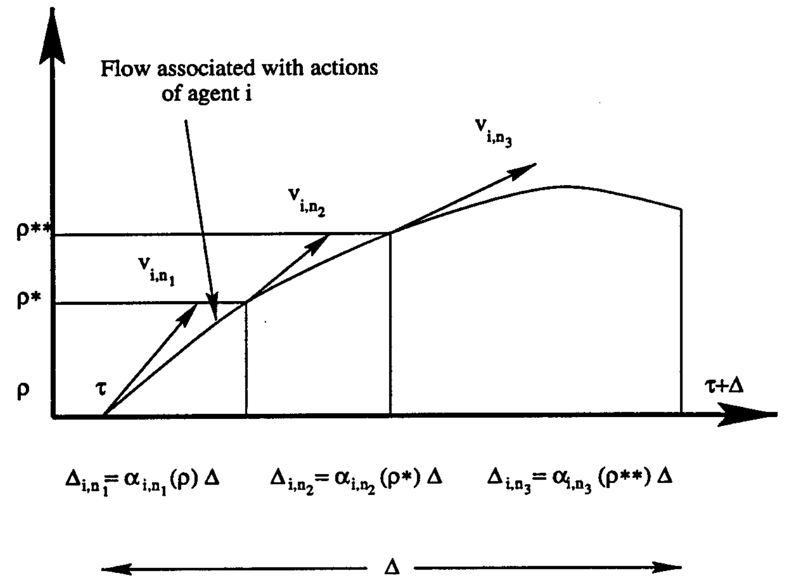

Flow associated with actions of agent i
v_{i,n_3}
v_{i,n_2}
v_{i,n_1}
ρ**
ρ*
ρ
τ
τ+Δ
$\Delta_{i,n_1} = \alpha_{i,n_1}(\rho)\,\Delta$
$\Delta_{i,n_2} = \alpha_{i,n_2}(\rho^*)\,\Delta$
$\Delta_{i,n_3} = \alpha_{i,n_3}(\rho^{**})\,\Delta$
Δ

Figure 5: Conceptual illustration of agent action schedule

and

$$\frac{d}{dt}\Psi(t,p) = v\mid_{\Psi(t,p)} \qquad \text{(flow generation)}$$

Now we are ready to customize these concepts for our model. Let $\Delta > 0$ be the width of the current decision interval, $[t, t+\Delta)$. Let $C_i^u(p,t)$ be the i-th unsatisfied demand at the beginning of the interval. Agent A_i has a set of primitive actions:

$$\{v_{i,j} : j = 1, \ldots, n_i \text{ where } v_{i,j}|_p \in TM_p \text{ for each } p \in m\} \tag{13}$$

During the interval $[t, t+\Delta)$, agent A_i schedules one or more of these actions to produce a flow which will bring the system closer to the goal. In particular, A_i determines the fraction $\alpha_{i,j}(p,t)$ of Δ that action $v_{i,j}$ must be executed as a function of the external perturbation functions $S_{r,i}(t,p)$ and the vector of the demand functions of the agents in the network $D(p,t) = (D_1(p,t), \ldots, D_{N(t)})$. Figure 5 conceptually illustrates a schedule of actions involving three primitives. We will use this example as means for describing the derivation of our model. The general case is similar.

The flow Ψ_i associated with the schedule of Figure 5 can be computed from the flows associated with each of the actions:

$$\Psi_i(\tau,p) = \begin{cases} \Psi_{v_{i,n_1}}(\tau,p) \text{ if } t \le \tau \le t + \Delta_{i,n_1} \\ \Psi_{v_{i,n_2}}(\tau, \Psi_{v_{i,n_1}}(\tau,p)) \text{ if } t + \Delta_{i,n_1} \le \tau \le t + \Delta_{i,n_1} + \Delta_{i,n_2} \\ \Psi_{v_{i,n_3}}(\tau, \Psi_{v_{i,n_2}}(\tau, \Psi_{v_{i,n_1}}(\tau,p))) \\ \qquad \text{if } t + \Delta_{i,n_1} + \Delta_{i,n_2} \le \tau \le t + \Delta_{i,n_1} + \Delta_{i,n_2} + \Delta_{i,n_3} \end{cases} \tag{14}$$

where $\Delta = \Delta_{i,n_1} + \Delta_{i,n_2} + \Delta_{i,n_3}$ and $\alpha_{i,n_1} + \alpha_{i,n_2} + \alpha_{i,n_3} = 1$. We note that the flow Ψ_i given by (14) characterizes the evolution of the process as viewed by agent A_i. The vector field $v_i|_p$ associated with the flow Ψ_i is obtained by differentiation and the third identity in (12). This vector field applied at p is proportional to

$$[v_{i,n_1}, [v_{i,n_2}, v_{i,n_3}]] \tag{15}$$

where $[.,.]$ is the Lie bracket due to the parallelogram law, see [63]. The Lie bracket is defined as follows. Let v and w, be derivations on M and let $f : M \to R$ be any real valued smooth functions. The Lie bracket of v and w is the derivation defined by

$$[v, w](f) = v(w(f)) - w(v(f)),$$

see [20].

Thus the composite action $v_i|_p$ generated by the i-th agent to to control the process is a composition of the form of (15). Moreover from a version of the Chattering lemma and duality [30], we can show that this action can be expressed as a linear combination of the primitive actions available to the agent as follows.

$$\begin{aligned} [v_{i,n_1}, [v_{i,n_2}, v_{i,n_3}]] &= \sum_j \gamma_j^i(\alpha) v_{i,j} \\ \sum_j \gamma_j^i(\alpha) &= 1 \end{aligned} \tag{16}$$

with the coefficients γ_j^i determined by the fraction of time that each primitive action $v_{i,j}$ is used by agent i.

The effect of the field defined by the composite action $v_i|_p$ on any smooth function (equivalent function class) is computed by expanding the right hand side of (14) in a Lie-Taylor series [65]. In particular, we can express the change in the demand modification functions C_i^u due to the flow over the interval Δ in terms of $v_i|_p$. The evolution of the modified demand function C_i^u over the interval starting at point p is given by

$$C_i^u(t + \Delta, p'') = C_i^u(t, \Psi_i(t + \Delta, p)) \tag{17}$$

Expanding the right hand side of (17) in a Lie-Taylor series around (t, p), we obtain,

$$C_i^u(t + \Delta, p'') = \sum_j \frac{(v_i|_p(C_i^u(p,t)))^j \Delta^j}{j!}$$

where

$$(v_i|_p(\cdot))^j = v_i|_p\left((v_i|_p(\cdot))^{j-1}\right) \tag{18}$$

and

$$(v_i|_p)^0(f) = f \text{ for all } f.$$

In general, the series in the right hand side of (18) will have countable many non-zero terms. In our case, since the topology of M is finite due to the fact that it is generated by finitely many logic clauses, this series will have only finitely many non-zero terms. Intuitively, this is so because in computing powers of derivations (i.e., limits of differences), we need only to distinguish among different neighboring points. In our formulation of the topology of M, this can only be imposed by the information in the clauses of the agent's Knowledge Base. Since each agent's knowledge base has only finitely many clauses, there is a term in the expansion of the series in which the power of the derivation goes to zero. This is important because it allows the series in the right-hand side to be effectively generated by a locally finite automaton. We will expand on the construction of this automaton in the next section when we discuss the inference procedure carried out by each agent.

We note that given the set of primitive actions available to each agent, the composite action is determined by the vector of fraction functions α_j. We will see in the next section that this vector is inferred by each agent from the proof of existence of solutions of an optimization problem.

Now we can write the specific nature of the model formulated in expression (4). At time t and at point $p \in M$ the demand modification function of agent i is given by :

$$C_i^u(p,t) = C_i^u(p,t^-) + S_{r,i}(p,t) + \sum_k Q_{i,k} D_k(p,t^-) \tag{19}$$

where t^- is the end point of the previous update interval, $S_{r,i}$ is the external perturbation function to agent i, and $Q_{i,k}$ is a multiplier determining how much of the current demand modification requirements of agent A_k is allocated to agent A_i. This allocation is determined from the characteristics of the process both agents are controlling and from the process description encoded in the agent's knowledge base. The actual request for service from agent k to agent i is thus the term, $Q_{i,k}D_k(p,t^-)$. The information sent to agent i by agent k is the demand modification function $D_k(p,t^-)$ at the end of the previous interval. Finally the point $p \in M$ carries the current status of the process monitored by the agents appearing in (19). Agent k thus contributes to Agent i's new control only if $Q_{i,k} \neq 0$.

This concludes our description of the model. For space considerations, some details have been left out. In particular those related to the strategy for activation and deactivation of agents. These will be discussed in a future paper. Now, we proceed to describe the architecture that the agents use to control the model presented in this section.

3 Process Control Architecture

In this section, we describe the main operational and functional characteristics of the DIS controller. As we mentioned in the introduction, this controller is implemented as a distributed system composed of agents and a communication network. The Multiple Agent Hybrid Control Architecture, MAHCA, operates as an on-line, distributed, plan generator and executor. At any update time t, each active agent has a "plan", an equational representation of the actions necessary to move towards a desired state in the next time slice Δt. The agent "proves" the plan is consistent with respect to the current system status as stored in it's knowledge base. As a side effect of proving that the plan is realizable, it generates control actions. That is, it generates the desired closed loop behavior of the logic communications network as viewed by the agent. The conjunction of plans at each instant of time, encodes the desired behavior of the entire network.

With respect to functionality, the agents in MAHCA in a DIS application are of two types: simulation agents (Figure 7) and demand agents (Figure 6). Each simulation agent operates as an intelligent interface to a simulation on the network. Each demand agent acts as an intelligent interface to a user in the DIS system. The actions generated by a simulation agents are directives to the simulation it interfaces with and the actions of a demand agent implement directives to the network that realizes the goals of the user interfaced to it. A MAHCA realization of a DIS Controller is a network of demand (user) agents and simulation agents and is shown in Figure 1.

Each agent of MAHCA, either a demand agent or a simulation agent, includes a core of six generic modules: a Planner, a dynamic Knowledge Base, a deductive Inferencer, an Adapter, a Knowledge Decoder and a Persistent Object Store System [30]. We briefly overview the functionality of these core modules.

The Planner generates a statement representing the desired closed loop behavior of the system as viewed by the agent. This statement is the plan proposed by the agent for a simulation or a user. The plan encodes the desired behavior corresponding to the user scenario goal for a demand agent or the desired simulation behavior for a simulation agent.

The Knowledge Base stores the requirements of operations or processes controlled by the agent. It also encodes system constraints, inter-agent protocols and constraints, sensory data, operational and logic principles and a set of primitive inference operations defined in the domain of equational terms.

The Inferencer determines whether the plan is realizable given the current status of the Knowledge Base. If the plan logically follows from the current status of the Knowledge Base, the Inferencer generates, as a side effect of proving the realizability of the plan, the current command actions. If the behavior statement does not logically follow from the current status of the Knowledge Base, that is, the plan is not realizable, the inferencer transmits the failed terms to the Adapter module for replacement or modification.

A Persistent Object Store System is the means of interfacing between an agent and the the network controller and scheduler to implement the network of cooperating agents (see Figures 6 and 7). This system has a basis of generic simulation trajectory primitives that are instantiated during run time to values determined by simulation data and stitched together to form a trajectory corresponding to the behavior of the simulations. The chief advantage of having this system is that it reduces the amount of state data that has to be transmitted between agents to construct the simulation trajectories.

Finally, the Knowledge Decoder translates knowledge data from the other agents, stored in the Persistent Object Store System, and incorporates them into the Knowledge Base of the agent.

In each agent of MAHCA, the plan is the formulation of a relaxed variational optimization problem whose successful resolution produces an action schedule of the form of (15). Each agent operates as a real-time solver of an appropriate relaxed variational optimization problem [26]. A customized version of this theory, enriched with elements of differential geometry, equational logic and automata theory provides a general representation for the dynamics, constraints, requirements and logic of the logic communication network. We devote the rest of this section to the discussion of the main elements of this theory in the context of the operational features of MAHCA.

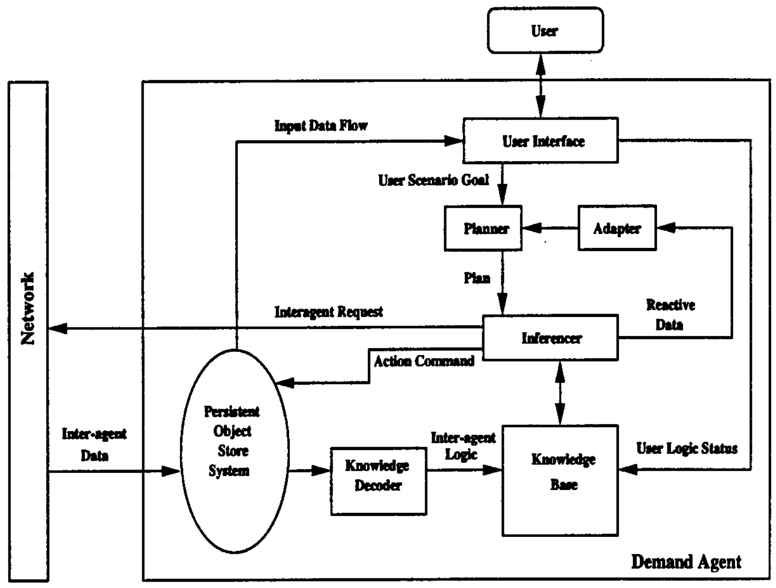


Figure 6: Demand Agent

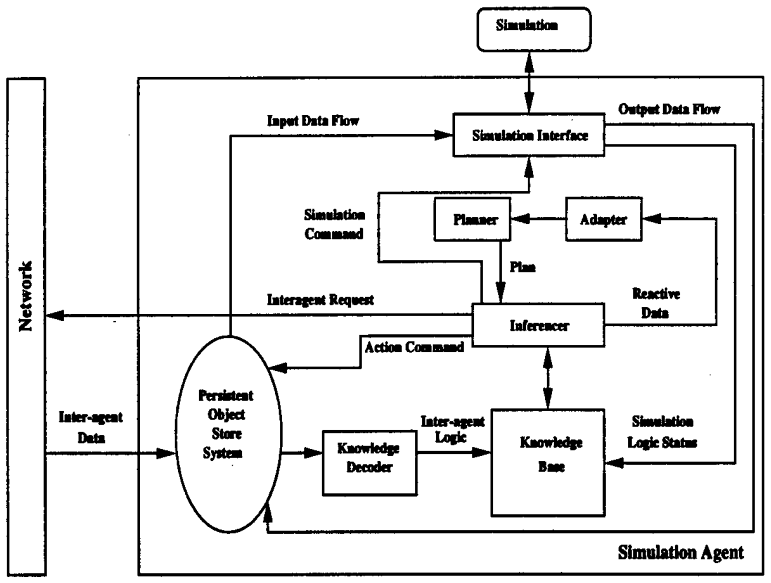


Figure 7: Simulation Agent

3.1 Core Architectural Elements of a MAHCAagent

We will discuss next the functionality of the six core modules of a MAHCA agent in a DIS application.
3.1.1. Knowledge Base: The Knowledge Base consists of a set of equational first order logic clauses with second order extensions. The syntax of clauses is similar to the ones in the Prolog language. Each clause is of the form

$$\mathit{Head} \leftarrow \mathit{Body} \tag{20}$$

where *Head* is a functional form, $p(x_1, ...x_n)$, taking values in the binary set $[true, false]$ with $x_1, x_2, \ldots, x_n$ variables or parameters in the domain M of the MAHCA network. The symbol $\leftarrow$ stands for logical implication. The variables appearing in the clause head are assumed to be universally quantified. The *Body* of a clause is a conjunction of one or more logical terms

$$e_1 \wedge e_2 \wedge \ldots \wedge e_n \tag{21}$$

where $\wedge$ is the logical 'and'. Each term in (21) is a relational form. A relational form is one of the following: an equational form, an inequational form, a covering form, or a clause head. The logical value of each of these forms is either true or false. A relational form e_i is true precisely at the set of tuples of values S_i of the domain taken by the variables where the relational form is satisfied and is false for the complement of that set. Thus for $e_i = e_i(x_1, \ldots, x_n)$, S_i is the possibly empty subset of M^n ,

$$S_i = \{(x_1, \ldots, x_n) \in M^n : e_i(x_1, \ldots, x_n) = true\}$$

so that

$$e_i(x_1, \ldots, x_n) = false \text{ if } (x_1, \ldots, x_n) \in M^n / S_i.$$

The generic structure of a relational form is given in Table 1.

Form	Structure	Meaning
equational	$w(x_1, \ldots, x_n) = v(x_1, \ldots, x_n)$	equal
inequational	$w(x_1, \ldots, x_n) \neq v(x_1, \ldots, x_n)$	not equal
covering	$w(x_1, \ldots, x_n) < v(x_1, \ldots, x_n)$	partial order
clause head	$q(x_1, \ldots, x_n)$	recursion,chaining

Table 1. Structure of the Relational Form

In Table 1, w and v are polynomic forms with respect to a finite set of operations whose definitional and property axioms are included in the Knowledge Base. A polynomic form v is an object of the form $v(x_1, \ldots, x_n) = \sum_{\omega \in \Omega}(v, \omega) \cdot \omega$ where Ω^* is the free monoid generated by the variable symbols $\{x_1, \ldots, x_n\}$ under juxtaposition. The term (v, ω) is called the coefficient of v at ω. The coefficients of a polynomic form v take values in the domain of definition of the clauses. The domain in which the variables in a clause head take values is the manifold M described in section 2. The logical interpretation of (20) and (21) is that the *Head* is true if the conjunction of the terms of *Body* are jointly true for instances of the variables in the clause head. These clauses, which are application dependent, encode the requirements on the closed-loop behavior of the model of the agent. In fact the closed loop behavior, which we will define later in this section in terms of a variational formulation, is characterized by continuous curves with values in M. This continuity condition is central because it is equivalent to requiring the system to look for actions that make the closed loop behavior satisfy the requirements of the plant model.

The denotational semantics of each clause in the knowledge base is one of the following:

1. a conservation principle,
2. an invariance principle, or
3. a constraint principle.

Conservation principles are one or more clauses about the balance of a particular process in the dynamics of the system or the computational resources. For instance, equation (19) encoded as a clause expresses the conservation of demand in the logic communications network.

$$
\begin{aligned}
&\mathit{conservation\_of\_unsatisfied\_demand}(p, t, [Q_{i,k}], S_{r,i}, [D_k], \Delta, C_i^u(t,p)) \leftarrow \\
&C_i^u(t+2\Delta, p'') = \sum_j \frac{(v_i|_p(C_i^u(t+\Delta, p')))\Delta^j}{j!} \ \wedge \\
&\qquad \text{/* encoding of equation (13) */} \\
&C_i^u(t+\Delta, p') = C_i^u(t,p) + S_{r,i}(t,p) + \sum_k Q_{i,k} B_k(t,p,\dot{p}) \ \wedge \\
&\qquad \text{/* encoding of equation (15) */} \\
&\mathit{process\_evolution}(p,t,p'') \ \wedge \ \text{/* encoding of equation (13) */} \\
&\mathit{consevation\_of\_unsatisfied\_demand}(p'', t+\Delta, [Q_{i,k}], S_{r,i}, [D_k], \Delta, C_i^u(t+2\Delta, p''))
\end{aligned}
\tag{22}
$$

In (22), the first equational term relates the unsatisfied demand for agent i at the current time to the unsatisfied demand in the past and the net current demand of the other agents connected to agent i. The last term of the rule implements the recursion.

As another example, consider the following clause representing conservation of computational resources:

$$
\begin{aligned}
\mathit{comp}(\mathit{Load}, \mathit{ProcessOp\_count}, \mathit{Limit}) \ \leftarrow \ & \mathit{process}(\mathit{process\_count}) \\
& \wedge\ \mathit{process\_count} \cdot \mathit{Load}_1 - \mathit{Op\_count} < \mathit{Load} \\
& \wedge\ \mathit{Load}_1 < \mathit{Limit} \\
& \wedge\ \mathit{comp}(\mathit{Load}_1, \mathit{Process}, \mathit{Op\_count}, \mathit{Limit}).
\end{aligned}
$$

where *Load* corresponds to the current computational burden, measured in VIPS (Variable Instantiations Per Second), *Process* is a clause considered for execution, and *Op_count* is the current number of terms in process.

Conservation principles always involve recursion whose scope is not necessarily a single clause, as in the example above, but with chaining throughout several clauses.

Invariance principles are one or more clauses establishing constants of motion in a general sense. These principles include stationarity, which plays a pivotal role in the formulation of the theorems

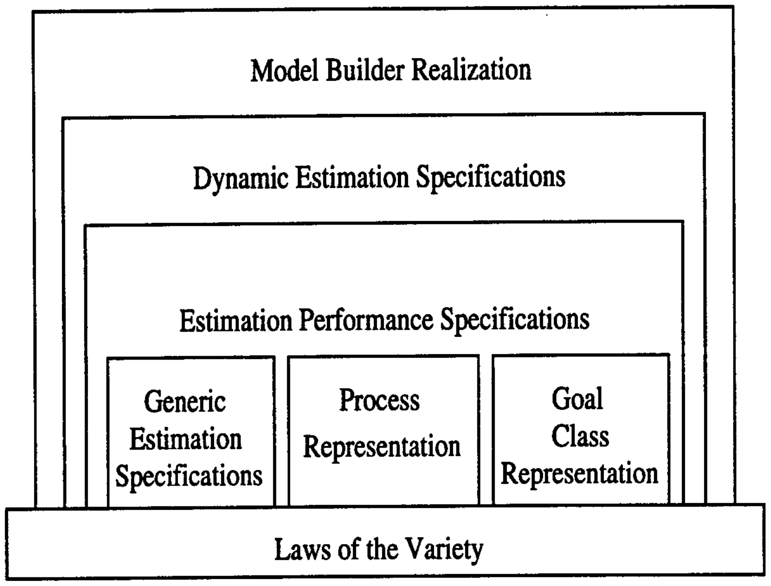

Model Builder Realization
Dynamic Estimation Specifications
Estimation Performance Specifications
Generic Estimation Specifications
Process Representation
Goal Class Representation
Laws of the Variety

Figure 8: Knowledge Base Organization

proved by the architecture, and geodesics. In the control of DIS processes, invariant principles specify quality response requirements. That is, levels of performance as a function of traffic load that the system must satisfy.

The importance of invariance principles lies in the reference they provide for the detection of unexpected events. For example, in a DIS process, the update time after a request is serviced is constant, under normal operating conditions. An equational clause that states this invariance has a ground form that is constant and any deviation from this value represents deviation from normality.

Constraint principles are clauses representing engineering limits to actuators or sensors and, most importantly, behavioral policies. For instance, in a DIS process, the characteristics of the speed of response of the values controlling the input flow size or the speed of access are given by empirical graphs (e.g. byte # vs. velocity with traffic volume as a parameter) stored in the system relational base. Another example in this application is given by the clauses that define the lifting strategy for embedding discrete varying trajectories into M (interpolation rules).

The clause database is organized in a nested hierarchical structure illustrated in Figure 8. The bottom of this hierarchy contains the equations that characterize the algebraic structure defining the terms of relational forms, i.e. an algebraic variety [65].

At the next level of the hierarchy, three types of clauses are stored: Generic Control Specifications, System Representation and Goal Class Representation.

The *Generic Control Specifications* are clauses expressing general desired behavior of the system. They include statements about stability, complexity and robustness that are generic to the class of declarative rational controllers implemented by MAHCA agents. These specifications are written by constructing clauses that combine laws of the kind which use the Horn clause format described earlier.

The *Process Representation* is given by clauses characterizing the dynamic behavior and structure of the plant, which includes sensors and actuators. These clauses are written as conservation principles for the dynamic behavior and as invariance principles for the structure. As in the case of Generic Control Specifications, they are constructed by combining a variety of laws in the equational Horn clause format.

The *Goal Class Representation* contains clauses characterizing sets of desirable operation points in the domain (points in the manifold M). These clauses are expressed as soft constraints; that is, constraints that can be violated for finite intervals of time. They express the ultimate purpose of the control agent but not its behavior over time.

The next level of the hierarchy involves the *Control Performance Specifications.* These are typically problem dependent criteria and constraints. They are written in equational Horn clause format. They include generic constraints such as speed and time of response, and qualitative properties of state trajectories [59].

Dynamic Control Specifications are equational Horn clauses whose bodies are modified as a function of the sensor and goal commands.

Finally, *Model Builder Realization* clauses constitute a recipe for building a procedural model (an automaton) for generating variable instantiation (unification) and for theorem proving.

3.1.2. The Planner:

The function of the theorem Planner, which is domain-specific, is to generate, for each update interval, a symbolic statement of the desired behavior of the system, as viewed, say by agent i, throughout the interval. The theorem statement that it generates has the following form.

Given a set of primitive actions there exists a control schedule $v_i|_p$ of the form (16) and a fraction function differential $d\alpha(\cdot)$ (Figure 5) in the control interval $[t, t+\Delta)$ such that $d\alpha(\cdot)$ minimizes the functional

$$\int_t^{t+\Delta} L_i(\Psi_i(\tau,p), v_i|_p(G(_i(\tau,p)))\,d\alpha(p,d\tau) \qquad (23)$$

subject to the following constraints:

$$\begin{aligned} g_i(S_i, \Psi_i(t+\Delta,p)) &= G_i(t,X_i) \\ &\quad \text{(local goal for the interval)}, \\ \sum_m Q_{i,m}(p,t)D_m(p,t) &= V_i(p,t) \qquad (24) \\ &\quad \text{(inter-agent constraint, see (19))} \\ &\text{and} \\ \int_t^{t+\Delta} d\alpha(p,d\tau) &= 1. \end{aligned}$$

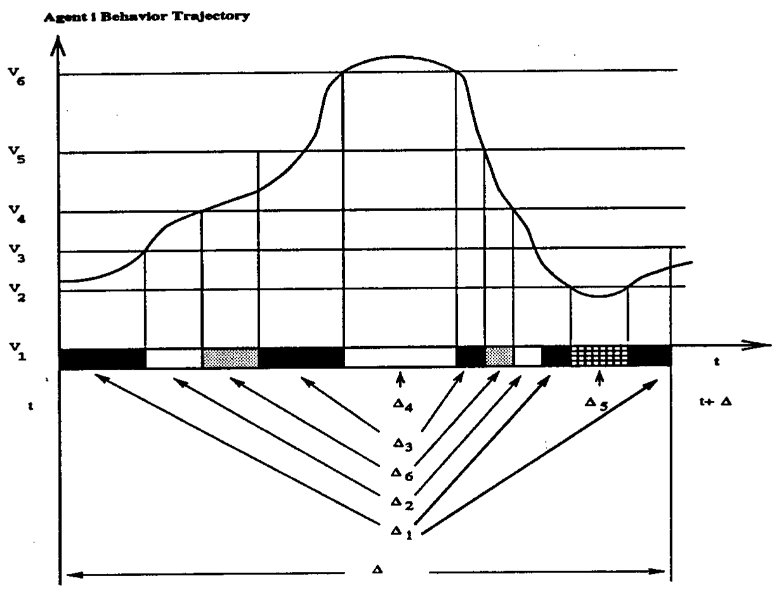


Figure 9: Illustration of optimization

In (23), L_i is the *Local Relaxed Lagrangian* of the system as viewed by Agent i for the current interval of control $[t, t+\Delta)$. This function, which maps the Cartesian product of the state and control spaces into the real line with the topology defined by the clauses in the knowledge base, captures the dynamics, constraints and requirements of the system as viewed by agent i. The relaxed Lagrangian function L_i is a continuous projection in the topology defined by the knowledge base (see [55]) in the coordinates of the i-th agent of the global Lagrangian function L that characterizes the system as a whole.

In (24), p represents the state of the process under control as viewed by the agent and G_i is the parallel transport operator bringing the goal to the current interval, see [41]. The operator G_i is constructed by lifting to the manifold the composite flow (see equation (14)). We note that the composite flow and the action schedule are determined once the fraction function is known and that this function is the result of the optimization (23), (24). In particular, the action schedule is constructed as a linear combination of primitive actions (see equation (16)).

The term $d\alpha(\cdot)$ in (23) is a Radon probability measure [65] on the set of primitive control actions or derivations that the agent can execute for the interval $[t, t+\Delta)$. It measures, for the interval, the percentage of time to be spent in each of the primitive derivations. The central function of the control agent is to determine this mixture of actions for each control interval. This function is carried out by each agent by inferring from the current status of its Knowledge Base whether a solution of the optimization problem stated by the current theorem exists, and, if so, to generate corresponding actions and state updates. Figure 9 illustrates the relations between the primitive actions and the fraction of Δ they are active in the interval $[t, t+\Delta)$.

The expressions in (24) constitute the constraints imposed in the relaxed optimization problem solved by the agent. The first one is the local goal constraint expressing the general value of the state at the end of the current interval. The second represents the constraints imposed on the agent by the other agents in the network. Finally, the third one indicates that is a probability measure. Under relaxation and with the appropriate selection of the domain, see [35], the optimization problem stated in (23) and (24) is a convex optimization problem. This is important because it guarantees that if a solution exists, it is unique up to probability, and also, it guarantees the computational effectiveness of the inference method that the agent uses for proving the theorem.

The construction of the theorem statement given by (23) and (24) is the central task carried out in the Planner. It characterizes the desired behavior of the process as viewed by the agent in the current interval so that its requirements are satisfied and the system "moves" towards its goal in an optimal manner.

3.1.3. Adapter:

The function under the integral in (23) includes a term, referred to as the "catch-all" potential, which is not associated with any clause in the Knowledge Base. Its function is to measure unmodeled dynamic events. This monitoring function is carried out by the Adapter which implements a generic commutator principle similar to the Lie bracket discussed in section 2. Under this principle, if the value of the catch-all potential is empty, the current theorem statement adequately models the status of the system. On the other hand, if the theorem fails, meaning that there is a mismatch between the current statement of the theorem and system status, the catch-all potential carries the equational terms of the theorem that caused the failure. These terms are negated and conjuncted together by the Inferencer according to the commutation principle (which is itself defined by equational clauses in the Knowledge Base) and stored in the Knowledge Base as an adaptation dynamic clause. The Adapter then generates a potential symbol, which is characterized by the adaptation clause and corresponding tuning constraints. This potential is added to criterion for the theorem characterizing the interval.

The new potential symbol and tuning constraints are sent to the Planner which generates a modified local Lagrangian for the agent and goal constraints. The new theorem, thus constructed, represents adapted behavior of the system. This is the essence of reactive structural adaptation in the our model

At this point, we pause in our description to address the issue of robustness. To a large extent, the adapter mechanism of each MAHCA agent provides the system with a generic and computationally effective means to recover from failures or unpredictable events. Theorem failures are symptoms of mismatches between what the agent thinks the system looks like and what it really looks like. The adaptation clause incorporates knowledge into the agent's Knowledge Base which represents a recovery strategy. The Inferencer, discussed next, effects this strategy as part of its normal operation.

3.1.4. Inferencer:

The Inferencer is an on-line equational theorem prover. The class of theorems it can prove are represented by statements of the form of (20) and (21), expressed by an existentially quantified conjunction of equational terms of the form:

$$\exists Z(W_1(Z,p)\ rel_i\ V_1(Z,p) \wedge \ldots \wedge W_n(Z,p)\ rel_i\ V_n(Z,p)) \tag{25}$$

where Z is a tuple of variables each taking values in the domain D, p is a list of parameters in D, and $\{W_i, V_i\}$ are polynomial terms in the semiring polynomial algebra,

$$\widetilde{D}\langle\Omega\rangle, \tag{26}$$

with $\widetilde{D} = (D, \langle +, \cdot, 1, 0\rangle)$ a semiring algebra with additive unit 0 and multiplicative unit 1. In (25), rel_i, $i = 1, \ldots, n$ are binary relations on the polynomial algebra. Each rel_i can be either an equality relation ($=$), inequality relation ($\neq$), or a partial order relation ($<$). In a given theorem, more than one partial order relation may appear. In each theorem, at least one of the terms is a partial order relation that defines a complete lattice on the algebra that corresponds to the optimization problem. This lattice has a minimum element if the optimization problem has a minimum. Given a theorem statement of the form of (25) and a knowledge base of equational clauses, the inferencer determines whether the statement logically follows from the clauses in the Knowledge Base, and if so, as a side effect of the proof, generates a non-empty subset of tuples with entries in M giving values to Z. These entries determine the agent's actions. Thus, a side effect is instantiation of the agent's decision variables. In (26), Ω is a set of primitive unary operations, $\{v_i\}$, the infinitesimal primitive fields defined in section 2. Each v_i maps the semiring algebra, whose members are power series involving the composition of operators, on Z to itself,

$$v_i : \widetilde{D}\langle\langle Z\rangle\rangle \rightarrow \widetilde{D}\langle\langle Z\rangle\rangle. \tag{27}$$

These operators are characterized by axioms in the Knowledge Base and are process dependent. In formal logic, the implemented inference principle can be stated as follows. Let Σ be the set of clauses in the Knowledge Base. Let $\Rightarrow$ represent implication. Proving the theorem means to show that it logically follows from Σ, i.e.

$$\Sigma \Rightarrow \textit{theorem}. \tag{28}$$

The proof is accomplished by sequences of applications of the following inference axioms:

(i) equality axioms

(ii) inequality axioms

(iii) partial order axioms

(iv) compatibility axioms

(v) convergence axioms

(vi) knowledge base axioms

(vii) limit axioms

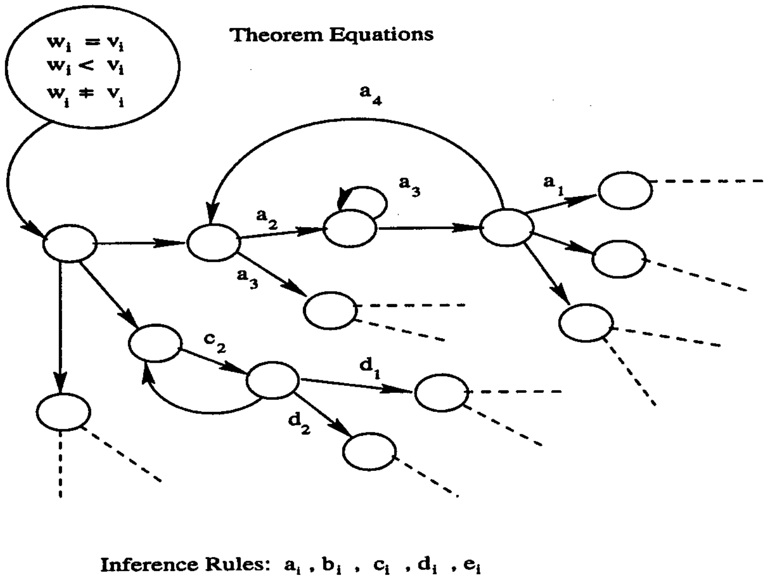


Figure 10: Conceptual Structure of the Proof Automaton

The specifics of these inference axioms can be found in [24] where it is shown that each of the inference principles can be expressed as an operator on the Cartesian product

$$\widetilde{D}\langle\langle W\rangle\rangle \times \widetilde{D}\langle\langle W\rangle\rangle. \tag{29}$$

Each inference operator transforms a relational term into another relational term. The inferencer applies sequences of inference operators on the equational terms of the theorem until these terms are reduced to either a set of ground equations of the form of (30) or it determines that no such ground form exists.

$$Z_i = \alpha_i, \alpha_i \in D. \tag{30}$$

The mechanism by which the inferencer carries out the procedure described above is performed by building a procedure for variable goal instantiation, i.e. a locally finite automaton. We refer to this automaton as the Proof Automaton. This important feature is unique to our approach. The proof procedure is customized to the particular theorem statement and Knowledge Base instance it is currently handling. The structure of the proof automaton generated by the inferencer is illustrated in Figure 10.

In Figure 10, the initial state represents the equations associated with the theorem. In general, each state corresponds to a derived equational form of the theorem through the application of a chain of inference operators to the initial state that is represented by the path

$$S_0 \xrightarrow{inf_1} S_1 \xrightarrow{inf_2} \cdots \xrightarrow{inf_k} S_k.$$

Each edge in the automaton corresponds to one of the possible inferences. A state is terminal if its equational form is a tautology, or it corresponds to a canonical form whose solution form is stored in the Knowledge Base. In traversing the automaton state graph, values or expressions are assigned to the variables. In a terminal state, the equational terms are all ground states, see (30). If the automaton contains at least one path starting in the initial state and ending in a terminal state, then the theorem is true with respect to the given Knowledge Base and the resulting variable instantiation is a valid one. If this is not the case, the theorem is false. The function of the complete partial order term present in the conjunction of each theorem provable by the inferencer is to provide a guide for constructing the proof automaton. This is done by transforming the equational terms of the theorem into a canonical fixed point equation, called the Kleene-Schutzenberger Equation (KSE) [24], which constitutes a blueprint for the construction of the proof automaton. This fixed point coincides with the solution of the optimization problem formulated in (23) (24), when it has a solution. The general form of KSE is

$$Z = E(p) \cdot Z + T(p) \tag{31}$$

In (31), E is a square matrix, with each entry a rational form constructed from the basis of inference operators described above, and T is a vector of equational forms from the Knowledge Base. Each non-empty entry, $E_{i,j}$, in E corresponds to the edge in the proof automaton connecting states i and j. The binary operator "$\cdot$" between $E(p)$ and Z represents the "apply inference to" operator. Terminal states are determined by the non-empty terms of T. The p terms are custom parameter values in the inference operator terms in $E(\cdot)$.

A summary of the procedure executed by the inferencer is presented in Figure 11.

We note that the construction of the automaton is carried out from the canonical equation and not by a non-deterministic application of the inference rules. This approach reduces the computational complexity of the canonical equation (low polynomic) and is far better than applying the inference rules directly (exponential).

The automaton is simulated to generate instances of the state, action and evaluation variables using an automaton decomposition procedure [61] which requires $n log_2 n$ time, where n is the number of states of the automaton. This "divide and conquer" procedure implements the recursive decomposition of the automaton into a cascade of parallel unitary (one initial and one terminal state) automata, see Figure 12. Each of the resulting automata on this decomposition is executed independently of the others. The behavior of the resulting network of automata is identical with the behavior obtained from the original automaton, but with feasible time complexity.

In Figure 12, the Parallel Decomposition transformation decomposes the finite state machine into a parallel series of unitary automata. A unitary automaton is a finite state machine with a single initial state and a single terminal state. The Cascade Decomposition then transforms each unitary automaton into a series of a prefix automaton followed by a loop automaton. A prefix automaton is a unitary automaton which has no transitions from its terminal state and a loop automaton is a unitary automaton whose initial state and terminal state coincide. The Linearization modifies the loop automaton by incorporating a new terminal state which has the same edges as the initial state of the loop automaton. Then the inaccessible states are trimmed from the resulting automata and the entire decomposition procedure is repeated if necessary. Whenever an automaton is produced in this decomposition which consist of a single path from the initial state to the terminal state, such

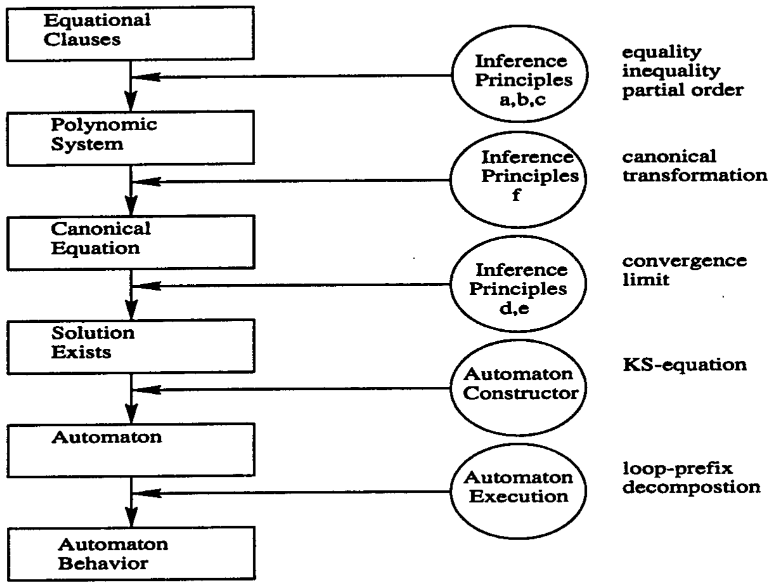


Figure 11: Summary of Inferencer Procedure

an automaton is called a path automaton, it corresponds to a successful path in the original finite state machine and an output is produced.

The $nlog_2n$ complexity of the procedure described above is a direct result of the non-deterministic nature of the procedure. The partial results of executing each path through the procedure are collected in the corresponding output states pictured at the bottom of Figure 12. The first path automaton in the decomposition produced by the procedure is executed and the result is stored in the corresponding output state. Once a successful path automaton is produced and executed, the decomposition procedure is terminated (first_finish_halt_strategy)

The Inferencer for each agent fulfills two functions,

(i) to generate a proof for the system behavior theorem of each agent generated by the Planner (equations (23) and (24)) and

(ii) to function as the central element in the Knowledge Decoder.

We now describe its function for proving the plan or behavior theorem. Later, we will overview its function as part of the Knowledge Decoder. To show how the inferencer is used to prove the Planner theorem, (23), (24), first, we show how this theorem is transformed into a pattern of the form of (25). Since (23), (24) formulates a convex optimization problem, a necessary and sufficient condition for optimality is provided by the following dynamic programming formulation.

$$V_i(Y,\tau) \quad = \quad inf_{\alpha_i} \int_T L_i(\Psi_i(\tau,Y), v_i|_p(G_i(\tau,p)))d\alpha_(p,d\tau) \tag{32}$$

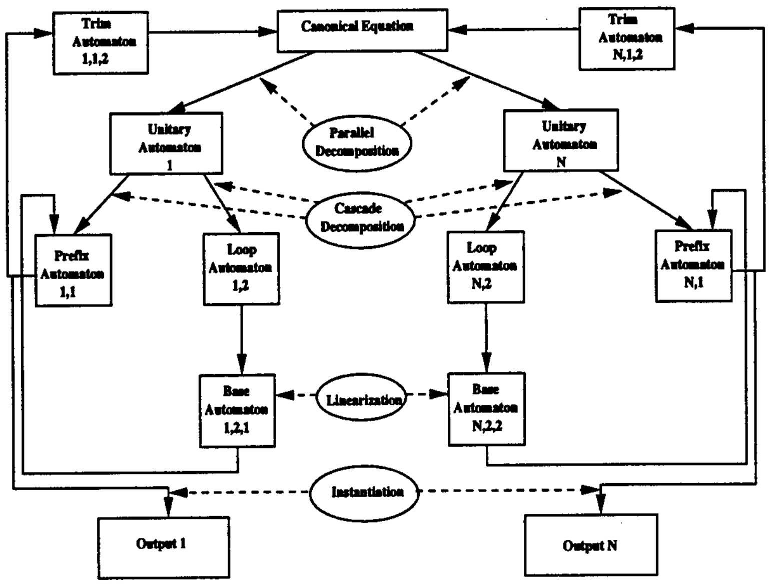


Figure 12: Automaton Decomposition Procedure

$$\begin{aligned} \frac{\partial V_i}{\partial \tau} &= inf_{\alpha_i} H(Y, \frac{\partial V_i}{\partial \tau}, \alpha_i) \\ &\text{where } Y(t) = p \text{ and } \tau \in [t, t+\Delta) \end{aligned}$$

In (32), the function V_i, called the optimal cost-to-go function, characterizes minimality starting from any arbitrary point inside the current interval. The second equation is the corresponding Hamilton-Jacobi-Bellman equation for the problem stated in (23) and (24) where H is the Hamiltonian of the relaxed problem. This formulation provides the formal coupling between deductive theorem proving and optimal control theory. The inferencer allows the real-time optimal solution of the formal control problem resulting in intelligent distributed real-time control of the multiple-agent system. The central idea for inferring a solution to (32) is to expand the cost-to-go function $V(.,.)$ in a rational power series V in the algebra:

$$\widetilde{D}\langle\langle (Y, \tau) \rangle\rangle \tag{33}$$

Replacing V for V_i in the second equation in (32), gives two items: a set of polynomic equations for the coefficients of V and a partial order expression for representing the optimality. Because of convexity and rationality of V, the number of equations to characterize the coefficients of V is finite. The resulting string of conjunctions of coefficient equations and the optimality partial order expression are in the form of (25). A detailed algorithmic approach to solving (32) which we call hybrid dynamic programming can be found in [42].

In summary, for each agent, the Inferencer operates according to the following procedure.

Step 1: Load current theorem (23), (24).

Step 2: Transform theorem to equational form (25) via (32).

Step 3: Execute proof according to Figure 11.

If the theorem logically follows from the Knowledge Base (i.e., it is true), the Inferencer procedure will terminate on step 3 with actions. If the theorem does not logically follow from the Knowledge Base, the Adapter is activated, and the theorem is modified by the theorem Planner according to the strategy outlined above. This mechanism is the essence of reactivity in the agent. Because of relaxation and convexity, this mechanism ensures that the controllable set of the domain is strictly larger than the mechanism without this correction strategy.

3.1.5 Knowledge Decoder:

The function of the Knowledge Decoder is to translate knowledge data from the network into the agent's Knowledge Base by updating the inter-agent specification clauses. These clauses characterize the second constraint in (32). Specifically, they express the constraints imposed by the rest of the network on each agent. They also characterize the global-to-local transformations (see [37]). Finally, they provide the rules for building generalized multipliers for incorporating the inter-agent constraints into a complete unconstrained criterion, which is then used to build the cost-to-go function in the first expression in (32). A generalized multiplier is an operator that transforms a constraint into a potential term. This potential is then incorporated into the original Lagrangian of the agent which now accounts explicitly for the constraint.

The Knowledge Decoder has a built-in inferencer used to infer the structure of the multiplier and transformations by a procedure similar to the one described for (14). Specifically, the multiplier and transformations are expanded in a rational power series in the algebra defined in (33). Then the necessary conditions for duality are used to determine the conjunctions of equational forms and a partial order expression needed to construct a theorem of the form of (25) whose proof generates a multiplier for adjoining the constraint to the Lagrangian of the agent as another potential.

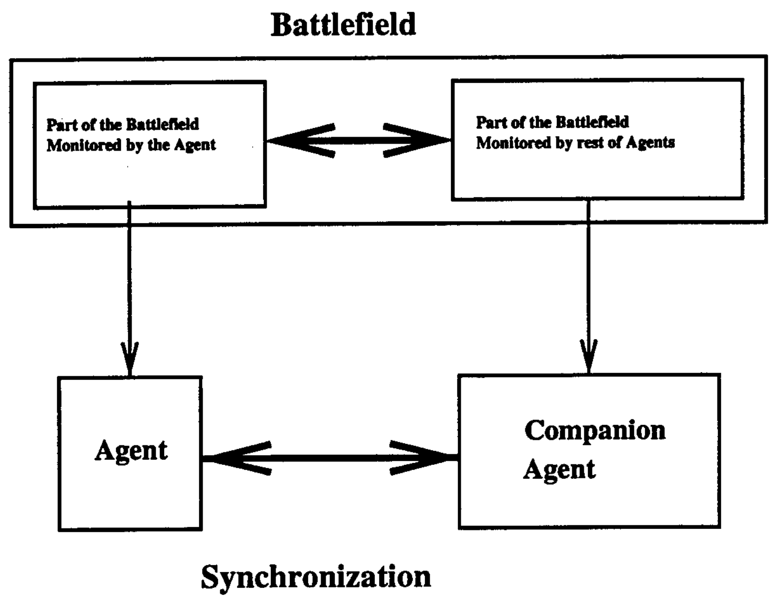
The conjunction of equational forms for each global-to-local transformation is constructed by applying the following invariant imbedding principle:

> For each agent, the actions at given time t in the current interval, as computed according to (32), are the same actions computed at t when the formulation is expanded to include the previous, current, and next intervals.

By transitivity and convexity of the criterion, the principle can be analytically extended to the entire horizon. The invariant imbedding equation has the same structure as the dynamic programming equation given in (32), but with the global criterion and global Hamiltonians instead of the corresponding local ones.

The local-to-global transformations are obtained by inverting the global-to-local transformations, obtained by expressing the invariant embedding equation as an equational theorem of the form of (25). These inverses exist because of convexity of the relaxed Lagrangian and the rationality of the power series.

It is important at this point to interpret the functionality of the Knowledge Decoder of each agent in terms of what it does. The multiplier described above has the effect of aggregating the



Battlefield
Part of the Battlefield Monitored by the Agent
Part of the Battlefield Monitored by rest of Agents
Agent
Companion Agent
Synchronization

Figure 13: The Companion Agent

rest of the system and the other agents into an equivalent companion system and companion agent, respectively, as viewed by the current agent. This is illustrated in Figure 13.

The aggregation model (Figure 13) describes how each agent perceives the rest of the network. This unique feature allows us to characterize the scalability of the architecture in a unique manner. Namely in order to determine computational complexity of an application, we have only to consider the agent with the highest complexity (i.e., the local agent with the most complex criterion) and its companion.

3.1.6 The Persistent Object Store

As described above, our Persistent Object Store System is the means of interfacing between an agent and the network controller and scheduler to implement the network of cooperating agents, see Figures 6 and 7. This system has a basis of generic simulation trajectory primitives that are instantiated during run time to values determined by simulation data and stitched together to form a trajectory corresponding to the behavior of the simulations. The goal of this system is to reduce the amount of state data that has to be transmitted between agents to construct the simulation trajectories.

Next we describe one of the key components in our approach: the use of light weight, high-performance data management as an enabling technology for DIS. DIS must balance computing, data management, and networking. High-performance computing has always played a central role in interactive simulations. On the other hand, the equally important companion role of high performance data management is only now emerging. High performance data management is important for DIS for several reasons.

1. Complex simulations produce large amounts of data, especially those running on high-performance computers. To minimize the total elapsed time of a simulation, it is important not only to reduce the processing requirements of the simulation, but also to reduce the time required to load data into the simulation, to store data produced by the simulation, and to move data between stages in the simulation. In other words, high performance data management can be equally effective as high performance computing at reducing the total elapsed time of a complex simulation.

2. The fact that a simulation is distributed provides the opportunity to optimize the simulation by not only moving the computational tasks to the data, as is traditional in simulations, but also, when appropriate, of moving data, or replicated data, to the tasks. More generally, distributed high performance data management provides the means to move the task to the data, the data to the task, or to use some hybrid strategy.

3. The specific goal of achieving a single integrated image of, say, a battlefield involving synthetic environments and realistic modeling and rendering of physical environments requires that large amounts of data be managed and distributed. High performance data management provides precisely the tools to do this.

Traditionally, scientific computing has faced a no-win situation: either develop and maintain a proprietary system for data management or use a general purpose database. In the past, the first approach could not exploit commercial software or standards, while the second approach could not provide the performance demanded by most simulations. Fortunately, recent advances in operating systems, programming languages, and compilers has provided a third approach: develop light weight software tools for high performance data management. By light weight, we mean that a limited functionality is provided, but with the benefit that the software tool itself can be relatively simple. The result is high-performance yet requires less effort for porting and maintenance.

Another key idea is to replace file-based access to data with object-based access. With object-based data access, applications such as DIS can easily exploit specialized application-specific algorithms for querying, storing and accessing data in order to obtain greater functionality and gain higher performance. In other words, rather than accessing unstructured data using the file system, one accesses the relevant structured objects by querying a persistent object store. Although the tables in a relational database are special types of objects, the type of scientific data occurring in simulations is typically of sufficient complexity that is usually more efficient to store and access it as complex objects, together with the appropriate pointers, than in the normalized form typical of relational databases. With both the data and processors distributed, agents perform a crucial function by connecting a request (or query) to the appropriate data and the appropriate resources.

There are other possible applications of persistent object stores in our DIS architecture. To understand these additions application, we need to consider the global structure of a DIS application. A distributed interactive simulation may be thought of as a hybrid system consisting of a distributed collection of dynamical systems and automata.

1. The dynamical systems with inputs and outputs model the physical components of the simulation, the rendering of the environment, and other phenomena modeled by continuous laws.

2. The automata with inputs and outputs model the software components of the simulation and select the appropriate dynamical systems for a particular time period.

An agent is a collection of variables, predicates and rules which manage processing and data requests. By updating variables and evaluating rules and predicates, our multiple agent architecture selects an automaton in the hybrid system to be replaced by a new automaton. That is, just as an automaton, which is internal to a hybrid system, selects an appropriate dynamical system, an agent, which is external to a hybrid system, selects an appropriate automaton. To summarize, we envision a distributed interactive simulation to also consist of a distributed collection of agents: The agents are rule-based systems which coordinate the different distributed components of the simulation by handling requests, monitoring the system components, and selecting appropriate automata to achieve a requested action.

Both automata and agents can be used to achieve a given behavior of a complex system by selecting one component from a collection of components. The differences are that automata operate on a fast clock by accepting an input symbol, transiting to a new state, and outputting a symbol which specifies the appropriate dynamical system, while agents operate on a slow clock by updating internal variables, recomputing predicates using rules, and selecting the appropriate actions determined by the rules.

Hybrid systems may be viewed from two complementary viewpoints: the discrete viewpoint in which the continuous system is viewed as a discrete collection of continuous objects, such as trajectory segments; and the continuous viewpoint, in which the discrete automaton is viewed as a continuous system with discrete transitions. From the discrete viewpoint, persistent object managers appear, for example, as a mechanism to manage physical collections of trajectories' objects. Once managed in this framework, it is a simple task to interface discrete digital automata to select trajectory segments with desired performance specifications or to provide appropriate control laws to the nonlinear input-output system, see [17] for further discussion. An example of this type of interaction is found in [19] which describes a path planning algorithm for hybrid systems. In this algorithm during preprocessing, large numbers of trajectory segments for a hybrid system are computed and used to populate a persistent object store of trajectory segments. Subsequently, a request to approximate a desired flight path is interpreted as a query on the persistent store of trajectory segments. The query returns a sequence of trajectory segments which approximate the desired flight path. Near real-time performance results from using appropriate indices. One strength of this approach is that the actual controls required can be retrieved along with the trajectory segments at no additional cost.

Another application of persistent object stores is to use persistent object store of predicates to implement certain function of the agent. That is, an agent also has discrete symbolic data. It is natural to use a persistent object to manage this data. More specifically, a low overhead, high performance object manager can be used to manage the distributed collection of predicates required by a system of cooperating autonomous agents in order to provide the performance required for real time applications. High performance inferencing could then be achieved by the use of a companion software tool which would access predicate rules and facts stored by the persistent object manager and provide high performance inferencing. Thus a "light weight" inferencer plus a distributed, persistent object manager can provide an alternative to Prolog or LISP for implementing distributed agents. For example, consider a DIS which uses multiple agents for the control of hybrid systems. The present implementation described in [40] for autonomous controlling agents uses compiled MetaProlog (Quintus Prolog) programs for each agent. The agents observe their local plant behavior and occasionally change the finite automaton controller of the plant when the performance specification is not met. With currently available commercial implementations, MetaProlog has serious known limitations when used for real-time control of large systems. There are two causes. One is the basic Prolog mechanism of backwards search. This one can be eliminated at some additional overhead by forward chaining implementations. But the other, more troublesome, cause is the necessity of retrieving and depositing rules and facts in large databases for on-line theorem proving in real-time. In the approach described in [17], the agent proves the theorem for a discretized optimization problem expressed in automaton equation terms that there is a control law that is sufficiently optimal for use for the next interval of time.

The crucial element here is to eliminate the bottleneck due to having to consult a large database to prove this theorem, that is, to choose a control law that is essentially obtained by back propagation (backwards chaining) from the final goal. These databases of rules and facts can be very large because the dimension and size of the state space of the controlled systems can be large. Prolog based systems are not very fast in performing this task, as the lack of commercial exploitation of deductive database systems over a twenty-five year period attests. General purpose databases and MetaProlog currently carry overhead that makes such large real-time applications hard to implement, necessitating disabling much of the Prolog mechanisms and using ad hoc database management techniques. Using a light weight persistent object manager together with a light weight inferencer to manage predicate data, stored rules and facts, as proposed in [17] appears to be a promising approach.

4 Architectural Elements of a Declarative Control Network

In this section, we shall discuss several important problems that a declaritive control network must overcome to function efficiently in a DIS architecture. In particular, we shall discuss the so-called phase coherence problem and it possible solution.

In MAHCA we synchronize state trajectories so that their evolution is continuous with respect to the topology defined by the stored knowledge. The only interagent synchronization mechanization MAHCA employs to achieve distributed simulation control is the result of the flow of equational logic terms between agents which alter the functional interaction of the agent with its simulator. A group of agents can exhibit causally inconsistent behavior due to the order of generation of events by two agents. An example case may be when a tank and helicopter are on opposing forces and are being simulated by different simulations. Suppose the tank has engaged an opponent in a house and the helicopter has engaged the tank. Suppose now that the result of the helicopter engagement is that the tank is destroyed. Because of latency, it may happen that the user agent is not aware of the tank destruction and it sets a goal for destruction of the house. Thus we have a loss of consistent local times between the agent controlling the helicopter simulation and the agent controlling the tank simulation. This is manifested by a phase incoherence, or discontinuity, between the phase of the state behavior of the user agents and of the simulation agents. As soon as this is detected by the user agent inferencer, this detection causes a logic constraint clause to fail. A side-effect of this logic failure is the identification of the offending terms in the user agent plan. Reactivity, which includes an interagent request, results in resolving the logic failure. We refer to this sequence of steps as restoring phase coherence or synchrony.

The inter-agent communication network's main function is to transfer inter-agent constraints among agents according to a protocol written in equational Horn clause language. These constraints include application dependent data and, most importantly, inter-agent synchronization. The inter-agent synchronization strategy is very simple. An agent is synchronous with respect to the network if its inter-agent constraint multiplier is continuous with respect to the current instance of its active knowledge. Since the equational Horn clause format allows for the effective test of continuity (which is implicitly carried out by the inferencer in the Knowledge Decoder), a failure in the Knowledge Decoder theorem results in a corrective action toward re-establishing continuity with respect to the topology defined by the current instance of the knowledge base [29].

The specification of the geometry of the network, as a function of time, is dictated primarily by global observability. By global observability, we mean the closure of the knowledge of the system as whole relative to the scope the systems reactivity. One of the central tasks in any application is to provide knowledge in the equational clause format to characterize global observability for the hybrid systems.

4.1 The Challenge of Communicating Systems of Equations Between Autonomous Objects:

One of the key problems for any battlefield DIS systems is minimize the information needed to communicate to give the location of objects such a tank, helicopters, etc.

The MAHCA DIS architecture consists of cooperating demand agents and simulation agents which add intelligence to the current DIS architecture, providing the ability to reconcile differences among its component constructive, live and virtual simulations. In this context, the intelligent controllers sit between the physical communications network and the simulation nodes, as shown in Figure 1. They perform intelligent interpretation of the contents of all Protocol Data Units (PDUs) sent and received. PDUs often must include analog models transmitted as sets of equations:

The overall principle to minimizing the communications bandwidth used to transmit information on the locations of simulation objects is that only state change information should be transmitted. Whenever a simulation object begins an activity which will continue for quite a while (where the definition of "quite a while" depends upon the viewpoint of the user of the simulation), the idea is to broadcast that the activity is beginning, and to then provide occasional updates. For example, conceptually one wants to transmit:

Event: start movement from A to B; and

Update: continuing movement from location C.

What is required is a type of "dead reckoning" strategy. That is, each node must maintain a model of its own objects that corresponds to the model(s) being used by all the other nodes and that it must continuously compare its current information with the approximations being used by the other nodes. Thus when a state update is transmitted, we must include not only the correct position and orientation but also the velocity vectors and other information that is required for the other node to compute a new approximation of the objects position. Thus dead reckoning strategies correspond to equations that model continuous motion through time and space. For example, suppose a tank platoon sets off down a road at 40 mph. The most obvious correction to the model that the tanks are moving in a straight line is to recognize that they are following a road, and roads are rarely straight. A more sophisticated correction would be to notice that there are huge craters in the road at various points, and the tanks must slow down to get past those obstacles. Corrections for logical events, like encountering craters, are probably best handled by the tank movement simulator, which in some cases may broadcast an immediate notification of the interruption and, in other cases, will provide implicit notification when the next periodic update to the tank columns actual location is broadcast. If dead reckoning were the only continuous process in time and space which had to be simulated, and if there were one best dead reckoning strategy, then the special case solutions which have already been implemented would be adequate. However, a variety of dead reckoning strategies will be needed, to adequately model the motion of various kinds of objects (motion of ships and aircraft is more accurately handled by these methods). Moreover, dead reckoning is just one of a very large set of processes which are best modeled by sets of equations which describe relationships in time and space.

Hybrid systems theory provides both a conceptual basis and a computational framework for describing and implementing systems which are best described by a combination of evolution equations and discrete logical events. Dead reckoning is just one of the processes which is best described by a combination of logical and evolution models that our proposed MAHCA DIS architecture is designed to integrate with other simulation models.

Transmitting even a single equation between heterogeneous computers can be a challenging task. There are software engineering issues related to standardization of notation, compilation versus interpretation, and the accuracy of floating point representations of real numbers. There are daunting configuration management issues in a very large distributed system, e.g., the trust that the particular versions running at a given moment are consistent. The declarative control architecture of MAHCA is the result of a fresh look at these problems and issues.

Consider the problem of providing timely updates of operational information to a joint task force commander of a light task force enroute from the Continental United States (CONUS) to insertion in an unsecured objective area. Suppose that the possible alternative courses of action for the joint task force depend upon alternative methods of crossing a water barrier currently spanned by a bridge. Then:

The operational scenario being executed by the task force commander will have been developed, shared, and updated by referring to military control symbols superimposed onto a map of the terrain in the area of operations as a set of overlays (e.g. objectives, unit boundaries, routes of advance, coordination measures, fire, communications, and medical support, barrier plan, transportation plan, ...);

Depending on the time and tools available and the size of the force, the operational scenario may have been rehearsed with staff and operational elements using sand tables, map exercises, and Advanced Distributed Simulations. Information pertaining to alternative courses of action will have been developed during preparation for the operation and will have been stored in a variety of formats at different levels of command;

Information available from commercial and military sources provides timely updates of weather information which, in turn, affects the trafficability of terrain, feasibility of use of various avenues of approach, and combat effectiveness of opposing forces;

Information available from national technical means provides timely updates of opposing force size and disposition; and

The task force commander and his staff need the current information on weather conditions, accurate maps of the operational area, updates of recent changes to terrain features which might not be reflected in the maps, and the dispositions of opposing and friendly forces to execute the task force operations plan upon arrival in the area of operations. The information currency and accuracy is critical, and it must result from the best electronic and human intelligence gathering means available (e.g., the latest satellite pass). This means that updates to the task force databases must occur before, while, and after the task force is in transit to the area of operations.

The problem of determining whether the bridge over a water obstacle is usable for the contemplated operation, and further, informing the joint task force commander and his staff in a timely manner exemplifies the issues in maintaining consistent views of the battlefield.. The usability of the bridge for light armor operations depends not only on its ability to carry the heaviest piece of task force equipment but also on the trafficability of the avenues of approach to the bridge and avenues of egress from the bridge. Fire support plans and barrier plans for operations around the bridge are based on known enemy dispositions and anticipated movements. Air defense plans are developed based on air avenues of approach into the area of operations with plans to provide point defense of the bridge during task force operations around the bridge. Task organization are designed to overmatch enemy ground and air capabilities in the area of operations. Logistics support plans (transportation, medical and personnel) are based on availability of the bridge for combat service support operations. Terrain and operational entity data change at different rates in different formats with different degrees of accuracy (trust). Data pertaining to use of the bridge by various task force elements will come at different rates from multiple sources (reports from coalition partners, updates from multi-spectral satellite imagery, sensor readings from temperature, pressure, acoustic, magnetic and other environmental sensors, reports from friendly forces, ...). Commanders at different echelons need different levels of detail of local conditions at different update rates in order to create and execute orders for synchronization of combat power.

Now consider the problem of using automation-aided operations to assist in the operational rehearsal (and training) of force components in a virtual environment that generates the most likely battlefield situation/scenario; and actual operational execution, with command and control oversight, that is able to quickly adjust mission plans to changing situations.

Let's begin with the observation that if the communications infrastructure is good enough for an Advanced Digital Simulation (ADS) which meets DIS requirements, then it is certainly good enough to model communications on the battlefield. To keep things simple, let's assume that ADS communications have adequate bandwidth and no perceivable delay. This assumption is consistent with the DIS vision principles of "ground (absolute) truth" and autonomy of simulation nodes. The problem we will discuss in this section is how to model the low bandwidth, queuing delays and service interruptions of actual battlefield communications in the ADS context. To make the discussion even more concrete, suppose we are focusing on receipt and transmission of digitized messages by a particular

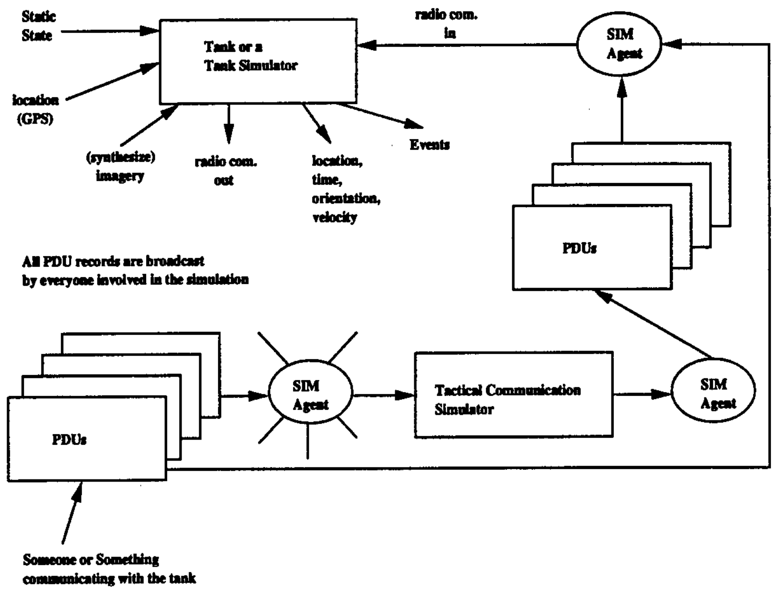

Static State
Tank or a Tank Simulator
radio com. in
SIM Agent
location (GPS)
(synthesize) imagery
radio com. out
location, time, orientation, velocity
Events
PDUs
All PDU records are broadcast by everyone involved in the simulation
SIM Agent
Tactical Communication Simulator
SIM Agent
PDUs
Someone or Something communicating with the tank

Figure 14: Possible (but inadequate) Simulation Architecture for Tactical Communications

tank. At the top of Figure 14 there is an idealized Tank Simulator which has a variety of inputs including static state information, location information (probably from GPS), (synthesized) imagery and tactical radio supporting both voice and data. Outputs of such a tank simulator might include location, time, orientation, velocity vector, specific events, and tactical radio. Focusing specifically on the tactical radio inputs and outputs, there will be a variety of communications partners, and the existence and quality of the communications links to each of those partners will depend on both the overall environment (e.g., the overall level of electro-magnetic activity in the region) and specific factors (objects breaking the line of sight between two partners, antenna directionality, etc.). The remainder of Figure 14 shows one possible architecture for inserting tactical communications into the DIS environment. In the center of the diagram is a tactical communications simulator inserted as a simulation object. By the principle of autonomy, the tank and all its communications partners can see all communications activity. It is up to the PDU interfaces to those simulators to ignore all "ground truth" PDU's, and to only see PDU's from the communications simulator. But this approach seems to violate the principle of autonomy of simulators because we have elevated the tank's *view* of tactical communications to the role of ground truth. This line of argument suggests that tactical communications must be treated in a similar manner to dead reckoning, with compatible models implemented in the PDU interfaces of every simulator.

However actual communications is orders of magnitude more complicated than dead reckoning. Individual simulators will model tactical communications at widely varying levels of abstraction, so the PDU interfaces must make appropriate translations. Levels of abstraction have proven useful for aggregation but a solution for disaggregation has yet to be implemented. For example, the tank simulator may want to receive the actual ASCII string to be displayed to the tank commander, but

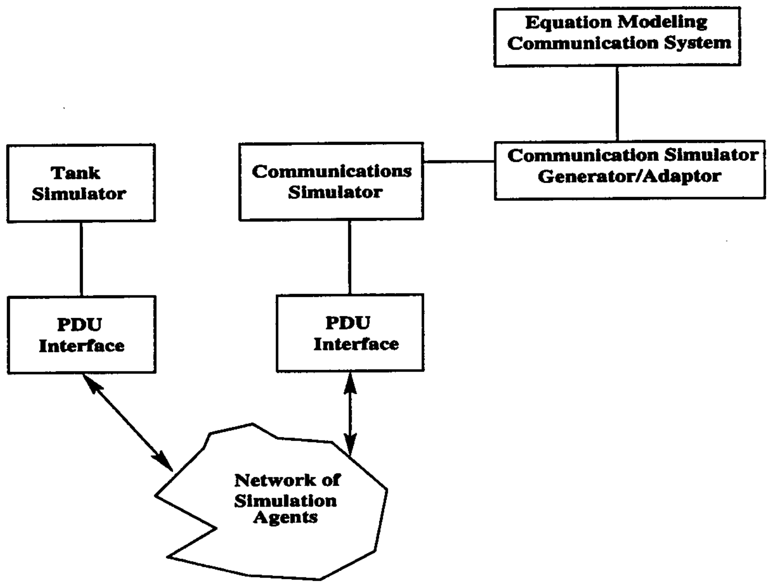

Equation Modeling Communication System
Tank Simulator
Communications Simulator
Communication Simulator Generator/Adaptor
PDU Interface
PDU Interface
Network of Simulation Agents

Figure 15: Declarative Control Architecture Approach to PDU Development

the real system may depend on the transmission of the same information in an encoded binary string sent using a certain secure radio protocol. The actual load on the communications net will depend on the encoding scheme, the protocol, and the communications media. The delivery time or failure to deliver will depend on priority, other traffic in the environment, and the positions of sender and receiver. While aggregation of message traffic at higher headquarters into sets of ASCII strings is useful for many purposes, disaggregation of the sets of strings into appropriate sets of the lower level forms (which determine operational loading of the communications network) is a very hard problem. This discussion leads to the model in Figure 15. The PDU interface to the tank simulator needs to share information with its simulation agent which tells it how the environment affects its ability to perceive radio signals in its environment. Such an agent could either provide a special entry to a special-purpose communications simulator running in the ADS environment, or it could provide a copy of a (standard) communications simulator which is integrated into the tank simulator's PDU interface.

In summary, MAHCA provides heretofore unavailable capabilities to the DIS community. Because models are "wrapped" with agents which implement formal analytical models of the simulation, new constraints and capabilities can be incorporated quickly. Because the executable simulation code is generated from the formal models, it is possible to meet efficiency requirements with code that is known to be correct. Because the MAHCA has already been demonstrated on similar problems, the approach is known to be practical. Because the approach makes full use of existing models, a powerful capability can be brought on-line quickly and the vision of realistic, shared, synthetic theaters of war will be much closer to fruition.

5 Battlefield Model Problem Domain: An overview of the target engagement problem

In this section, we shall provide an overview of target engagement scenario that was implemented via MAHCA agents at Odessey Research Associates as a demonstration project for the Army, see [34]. We are continuing to refine this demonstration at HyBrithms Coorporation. In this limited example, we shall be able to indicate the type of elements that are needed for a much larger demonstation of DIS applications.

We make no attempt to capture the complexity of the broad range of activities needed to protect the force or dominate the maneuver battle. Instead, we focus on a limited domain to demonstrate the power of a newly-developed technology to rigorously capture a substantial portion of the complexity of an important element of battle: the domain of target engagement. While our focus is limited, the complexity of engaging multiple targets by multiple weapons systems demonstrated the ability of the hybrid systems technology to reliably combine the logical constraints imposed by the commanders and crews of weapons platforms with the continuum constraints imposed by the physical environment to achieve an optimal response (as defined by the military commander) under uncertain conditions.

We emphasize that this limited demonstration is not intended to be a high-fidelity simulation of foe logic and dynamics, nor did it feature a high-fidelity reaction of friendly agents to foe activities. The demonstration does capture the fundamental logic and dynamics of engagement scenarios under the uncertainties of battle. We view this demonstration as the first experiment in a technology which supports a fundamentally new approach to incremental expansion of trusted systems.

Our specific objectives for the demonstration include the following:

1. Expand the generic architecture previously developed for multiple-agent hybrid control of distributed, real-time processes
2. Create a reference architecture for engagement processes
3. Demonstrate on-line rescheduling of real-time processes
4. Demonstrate parameter adaptation of architecture parameters in the presence of model parameter uncertainty
5. Demonstrate structural adaptation (i.e. automatic adaptation of the sharing of system control between structurally different models) in the presence of model structural uncertainty
6. Demonstrate use of the Equational Reactive Inferencer syntax as an appropriate architectural description language for simultaneously capturing the logic and the dynamics of the target engagement domain.

The model is discussed in Sections 5.1 through 5.6 below.

5.1 Battlefield Environment Model:

There are two major elements of the demonstration, (1) a Battlefield Environment, and (2) the controlling agents. We discuss these elements next.

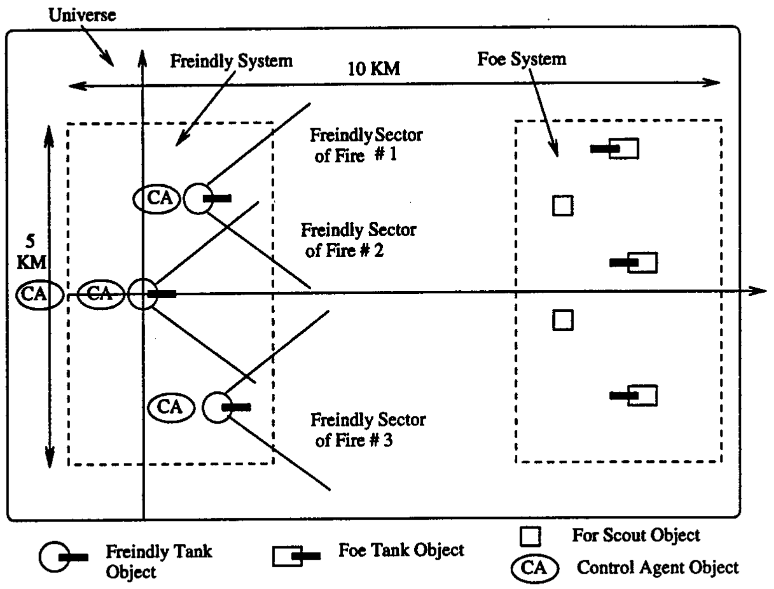


Figure 16: Battlefield Environment

The battlefield environment (see Figure 16) consists of a Universe (a prespecified region of the plane, i.e. a closed surface of R2), two types of objects, Friendly objects and Foe objects, and the rules which determine the reaction, i.e. the evolution of the state over time of friend or foe objects given the current state of friend and foe objects and the context of the operational situation which is a set of logical inputs. The set of possible battlefield scenarios is the set of sequences of friend and foe actions from initiation of the battlefield simulation until the friend and foe objects are in a terminal state and associated contexts.

In the implemented version of the demonstration, we had a limited set of friendly tank objects (always three objects) and foe objects (initially three tanks and two scouts) and a limited number of rules (equational clauses - see [44]) which determine the evolution of the state of the system. The model was deliberately constructed in order to reflect some of the actual conditions which occur on the battlefield but only at a level necessary to demonstrate the reasoning and behavioral capabilities of multiple-agent hybrid control. Specifically, we inserted a fourth control agent to provide supervisory control of the three tanks. Supervisory control actions were limited to assignment of sectors of fire and allocation of targets to engage. The actual decisions to fire are made by the local control agents for each tank. For purposes of the demonstration we simulated results of engagement decisions but in an actual system the program would be used in an advisory role.

Friendly command and control was limited to one commander (a Control Agent) making decisions based on battlefield sensors (Information Gatherer) and (noisy) information concerning friend and foe situation. Friendly weapons were fixed and consisted of direct fire weapons or smart mines (both labeled Attack System). Information concerning enemy activity was obtained from battlefield sensors aggregated into an agent labeled Information Gatherer. Thus, there are three friendly, hierarchically organized objects; two low-level objects each labeled Attack System and a high level object labeled Information Gatherer.

In our scenario, there were two types of enemy objects. Both types of enemy objects are crew-served vehicles. These vehicles are labeled Scout objects and direct-fire weapon systems (labeled Active objects) respectively. We allowed a variable number of foe objects (this number is specified at the beginning of a demonstration).

We will refer to the collection of three friendly objects as the Friendly System and the collection of foe objects as the Foe System.

The goal of the Friendly System is to coordinate the use of its assets (the three Attack systems. See Figure 11) to maximize the damage (Kill Probability) inflicted on the Foe System with admissible survivability (Survival Probability). The goal of the foe system is to detect the assets of the friendly system and to destroy them. The location of each of the Friendly objects in the coordinates of the universe is fixed. Each of the Foe objects follows prespecified (autonomous) trajectories in the universe. The friendly system acquires knowledge about position and perhaps the velocity of each of the foe objects via noisy sensors.

The friendly system is controlled by a MAHCA controller composed of three agents. Each of the objects in the friendly system, the two attack systems and the Information Gatherer are controlled by a MACHA agent. The battlefield scenario is illustrated in Figure 16.

We conclude this overview of our proposed model with a discussion of the script we are developing at HyBrithms Corporation for a future demonstration. The general idea is for the agents to implement on-line a coordinated reactive, adaptive policy [44] to 'win' the battle. This means achieving the highest kill probability over the prespecified duration of the script.

The actions of the control agent of each of the attack objects are of two types : tracking, via a noisy sensor, of a particular foe object, selected by the control agent of the Information Gatherer, and command firing against the foe object under tracking. The Information Gatherer decision domain includes an action to transfer the tracking action of a foe object to tracking another one. In an attack agent, the decision to engage a foe object is solely a function of the relation between the local kill and survival probabilities relative to the foe object under tracking and the global kill and survivabilities of the friendly system inferred by the Information Gatherer control agent.

5.2 Battlefield Environment (Engagement Scenario):

In this subsection, we summarize the engagement scenario.

1. Five targets at 5 locations going on 5 different paths:

   a) Tank on a crossing pattern,

   b) Scout on a diagonal, jinking pattern,

   c) Tank on a jinking, inbound pattern,

   d) Helicopter hovering and slow crossing, and

   e) Helicopter on a jinking pattern inbound and diagonal.

2. Three platforms at three fixed locations with three different sensor suites and capability suites:

  a) Tank with suite a,
  b) Tank with suite b, and
  c) Tank with suite c

3. The three platforms are controlled by three agents. The three agents are coordinated by a higher-level agent. One of the three agents obtains input from one of the platforms in the simulation world and sends appropriate commands to a single-agent controller which controls the test fixture.

4. An engagement scenario consists of:

   a) Targets moving on preassigned patterns. Some targets may be capable of detecting observation by active sensors. All targets will certainly detect engagement by friendly forces and may elect to engage or disengage.

   b) Each of the three platforms:

      (a) Detect targets using onboard sensors,
      (b) Identify targets,
      (c) Provide data to a controlling agent,
      (d) Select target to engage or are assigned a target by the coordinating agent,
      (e) Track selected target,
      (f) Engage target, and
      (g) Assess engagement and reengage or select another target.

5. Sensors need to be simulated in the battlefield environment. While data was provided for optical sensors, provision should be made in the data structure for other types of sensors:

   a) Optical including

      (a) Optical magnification,
      (b) Detection range,
      (c) Recognition range,
      (d) Identification range, and
      (e) Field of View,

   b) Thermal (data similar to optical),

   c) Millimeter wave (data similar to optical), abd

   d) Once target is detected, assume sensor provides azimuth and range information.

6. Nominal pointing parameters:

   a) Maximum acceleration,

   b) Maximum slew speed,

   c) Minimum smooth tracking slew speed (to avoid stiction),

d) Pointing accuracy,

e) Use nominal parameters to allocate error budget,

f) Data provided for ATAC, M1A1, Leopard, Centurion, and Apache.

7. Total system error data:

   a) Engagement accuracy can be divided into two elements: dispersion and bias.

   (i) Dispersion data accounts for differences in wind, ammunition, statistical deviations in system performance.

   (ii) Aiming bias can throw off engagement accuracy much more unexpectedly than aiming dispersion. Aiming bias increases with maneuvering target and distance to target (increased time of projectile flight).

8. We assume angular acceleration data influences aiming bias. We get angular acceleration data from tracker (an actual test fixture or the simulated sensor) but it is HARD to get accurate numbers for a maneuvering target (double the standard deviation for a non-maneuvering target).

9. We assume a circular target for calculation of probability of hit. We translate angular error data into a hit probability.

10. A clean interface between the multiple agents and the simulation of the platforms and the targets. This is so we can use the interface solution to drive other simulation software.

11. We assume that once a platform has fired, the subsequent hit probabilities go way down since the target will begin maneuvering.

12. The simulation begins with targets coming into the detection range of the sensors and ends with the attackers or defenders being eliminated.

5.2.1 Software Requirements: Demonstration Executive and Simulation Architecture:

In this section, we shall describe the modules which formed the basis of our demenstration.

The demonstration executive (Figure 17) consists of a set of initializations (Figure 18), a demonstration loop (Figure 19) which contains the simulation software and the MAHCA controller of the test fixture (hardware-in-the-loop), and data-recording executive. In our case, the test fixture was the ATB1000 of Picatinny Arsenal, see [48]. The ATB1000 is a test fixture designed by teh Army to simulate the types of non-linearities and disturbances that are present in a typical flexible beam stabization problem when firing a gun with a flexible barrel. The fixtures main body is an aluminum disk free to rotate in the horizontal plane. Affixed to the edge of the disk is an an interchangable steel rod (barrel) 1 meter in length.

Initialization (Figure 5.2.1) is accomplished for the simulation, controller and data recording processes.

The demonstration loop (Figure 19) object is the means for iterating through the demonstration cycle of simulation, data recording and safe termination.

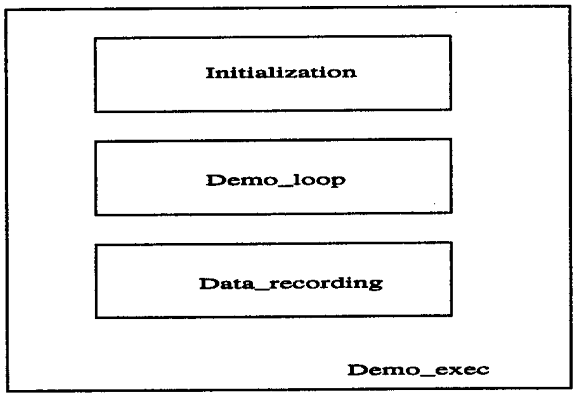


Figure 17: Demonstration Executive

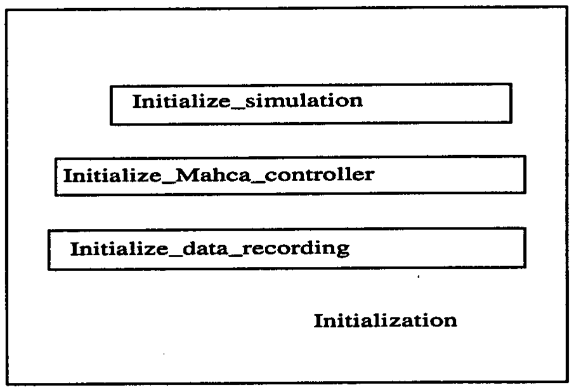


Figure 18: Initialization

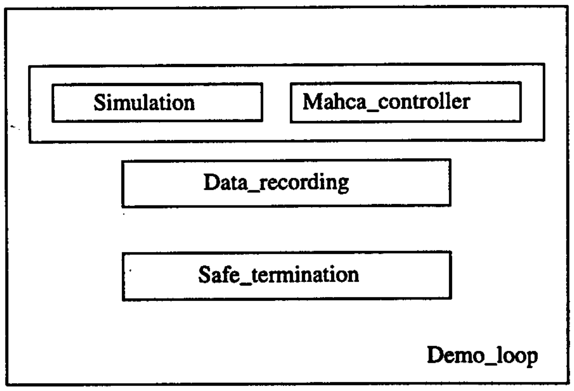


Figure 19: Demonstration Loop

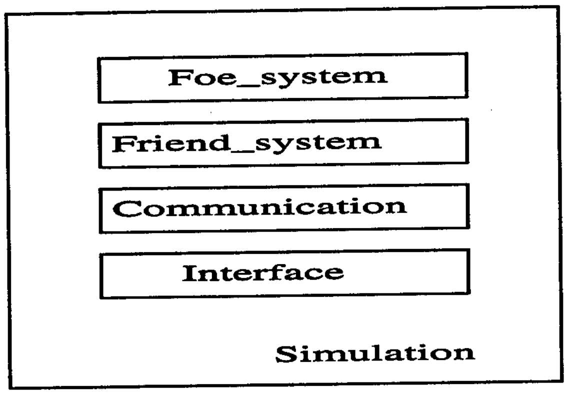


Figure 20: Simulation Architecture

The simulation object contains the basic elements of the engagement process (Figure 20). The semantics of the engagement process (as discussed in Section 5.2 above) defines interactions between the components of the simulation object. The simulation object contains descriptions of the foe system, the friend system, the communications system, and the interface definitions for passing information between these components. The interface requirements between the friend system and the foe system are such that the simulation of the foe system can be substantially changed without changing the friend system. We will develop the detailed models of the foe and friend systems in Sections 5.4 and 5.5.

5.3 Simulation Requirements for Foe System:

The foe system is described in the next six figures which describe six major components of the simulation of foe activities:

1. The top-level object of the foe system which contains the initialization logic for the individual foe objects, the logic for execution of foe activities, and the logic for elimination of foe objects from the simulation (Figure 21).
2. The foe eliminator which removes the foe object from the simulation upon determination by the foe system that this object has been destroyed (Figure 22).
3. The foe executor which contains the integration logic for each foe object and the switching logic which determines the logic mode switching for foe object activities (e.g. engage opposing force weapon system upon detection of firing action or flee from opposing force weapon system upon detection of firing action (Figure 23).
4. The object which initializes the foe objects, including the type of foe (e.g. normal description of type of tank, helicopter, or other weapon system), dynamics appropriate to the foe type, and logic appropriate to the foe type (Figure 24).

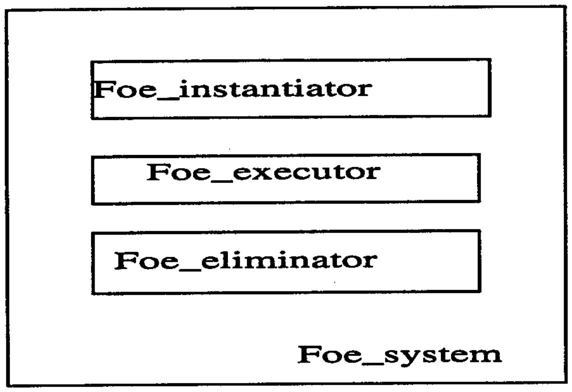


Figure 21: Foe System

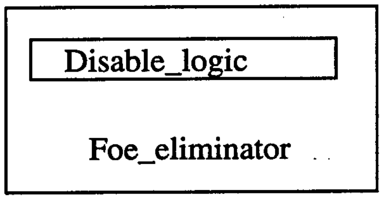


Figure 22: Elimination of Foe Objects

5. The foe object dynamics which contains the description of the object dynamics. Inputs to this object are the foe type, initial conditions (position, velocity, and acceleration), and acceleration model (for the different types of target tracks described in Section 2.1.3) (Figure 25).

6. The Foe object logic which includes detection of opposing force object range and engagement status and the firing logic (Figure 26).

5.4 Demonstration Model of Friend System:

The friend system is described in the next eight figures which provide an overview of friendly activities:

1. The top-level object of the friend system which contains the initialization logic for the individual friend objects, the logic for execution of friend activities, and the logic for elimination of friend objects from the demonstration (Figure 27).

2. The object which initializes the friend objects, including the type of friend (e.g. normal description of type of tank, helicopter, or other weapon system), dynamics appropriate to the friend type, logic for the agent controller, logic for the friend sensor, and firing logic appropriate to the friend type (Figure 28).

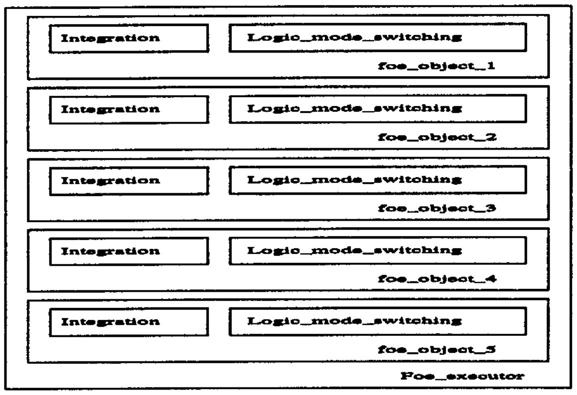


Figure 23: Execution of Foe Object

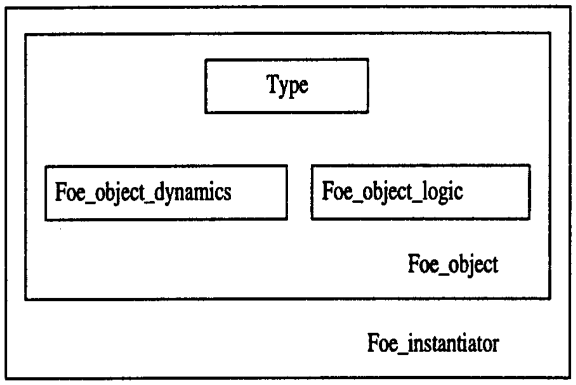


Figure 24: Initialization of Foe Objects

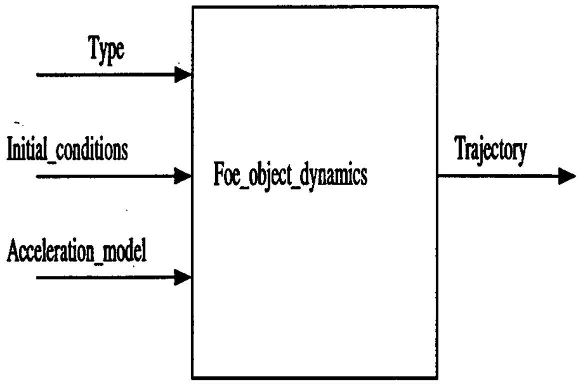


Figure 25: Implementation of Foe Object Dynamics

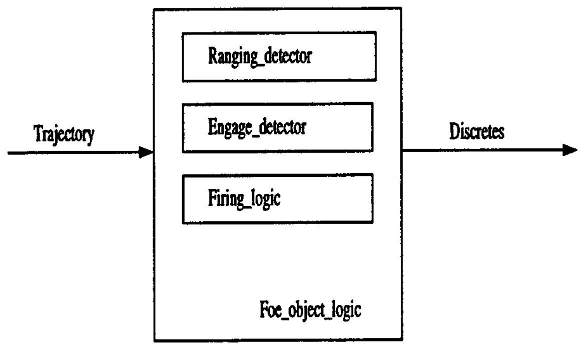


Figure 26: Foe Object Logic Requirements

3. The sensor object which combines the operational capabilities of several pieces of equipment to perform the functions of searching, detecting, recognizing, identifying, and tracking (Figure 29).
4. The friend object dynamics which receives initial conditions and other parameters and provides the object trajectory (Figure 30). For this demonstration the outputs of this object will be constant since we are keeping the friend objects stationary.
5. The firing logic model provides the basic logic for engagement "decisions" by the friendly agents (Figure 31).
6. The communication logic which contains the interface to the C and the application software, the application software, and the interface to network communication software (Figure 32).
7. The friend executor which instantiates each friend object (Figure 33).
8. A description of sample relational terms corresponding to some of the components of the demonstration (Figure 34).

5.5 System Implementation Overview

The system implementation of the demonstration is pictured in Figure 35. The Control agents of Friendly System was realized by three Intel 586 PC's, running Real Time DOS. The dynamics of the Foe System, one of the attack elements and the Information Gatherer are implemented as part of the environment simulation. Data recording, playback facilities and non real time processing are implemented on a 586-PC loaded with the Lab view environment.

Each of the control agents of the friendly system implements a MAHCA agent with its knowledge base populated with, in addition to the generic clauses, the clauses that express the dynamics of the kill and survivability probabilities.

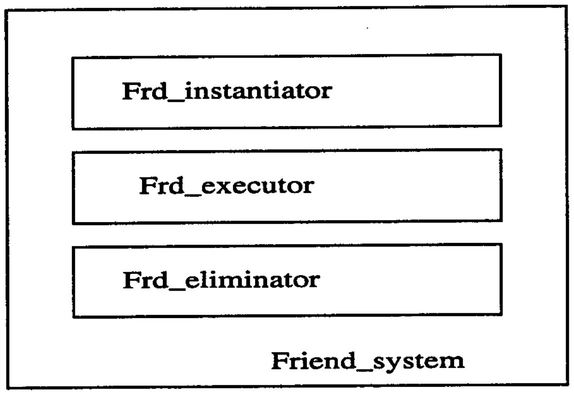


Figure 27: Friend System

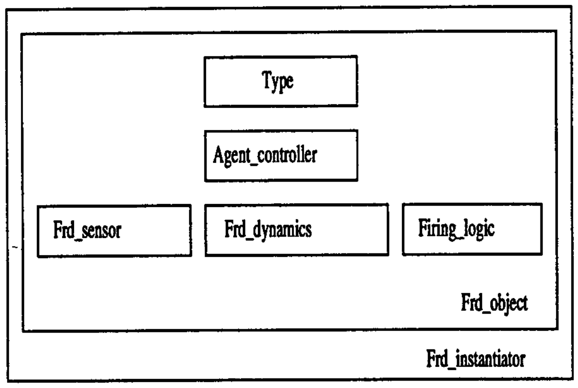


Figure 28: Friend System Instantiator

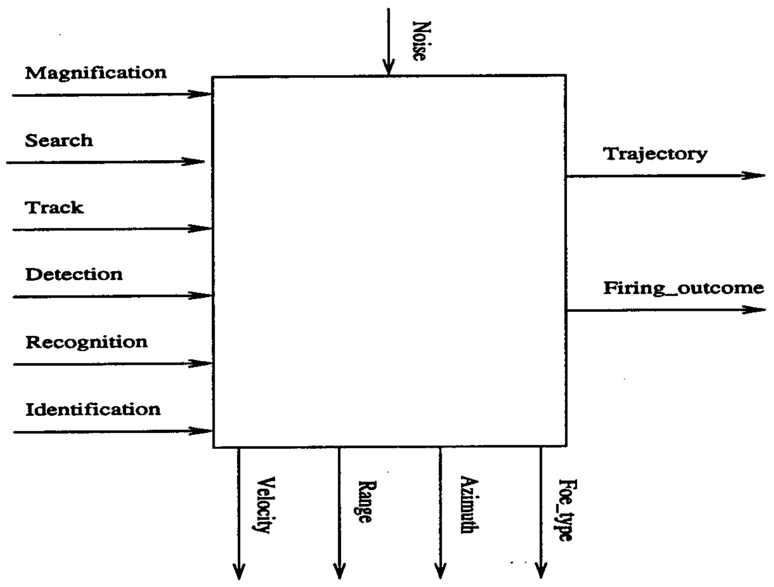


Figure 29: Sensor Object

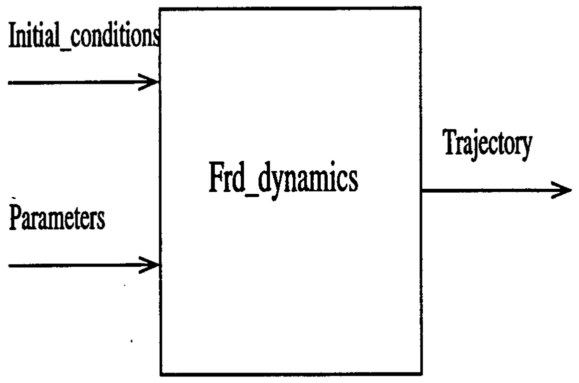


Figure 30: Friend Object Dynamics

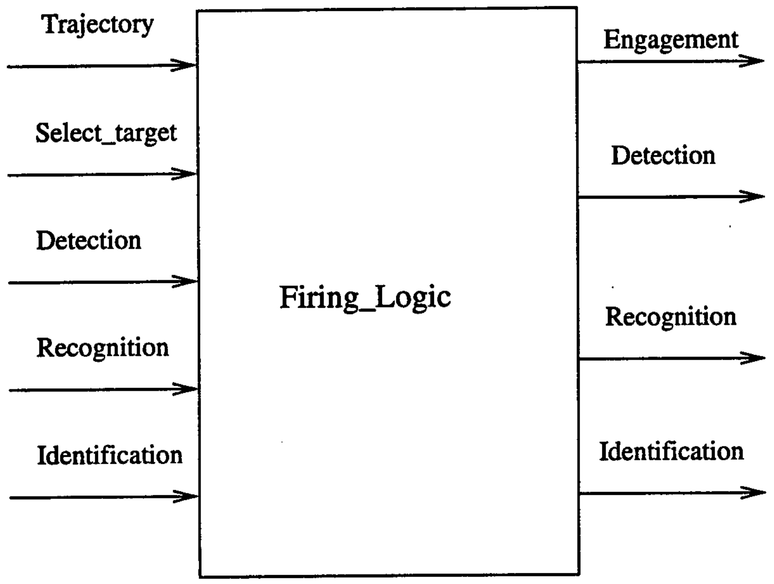


Figure 31: Firing Logic Model

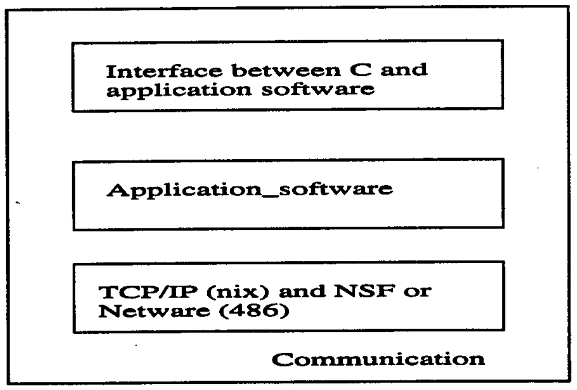

Interface between C and application software
Application_software
TCP/IP (nix) and NSF or Netware (486)
Communication

Figure 32: Communication Interfaces

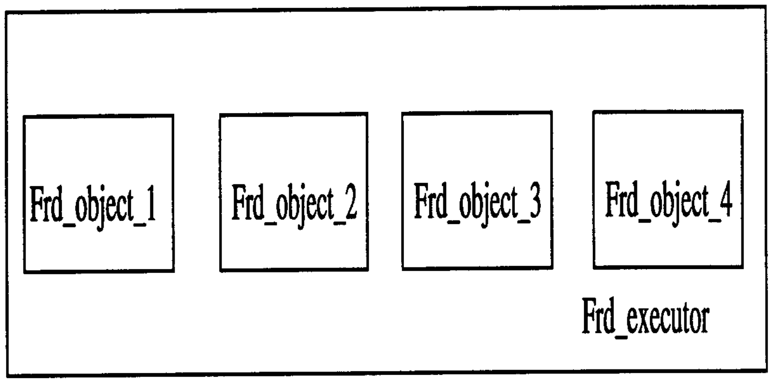


Figure 33: Friend Object Executor

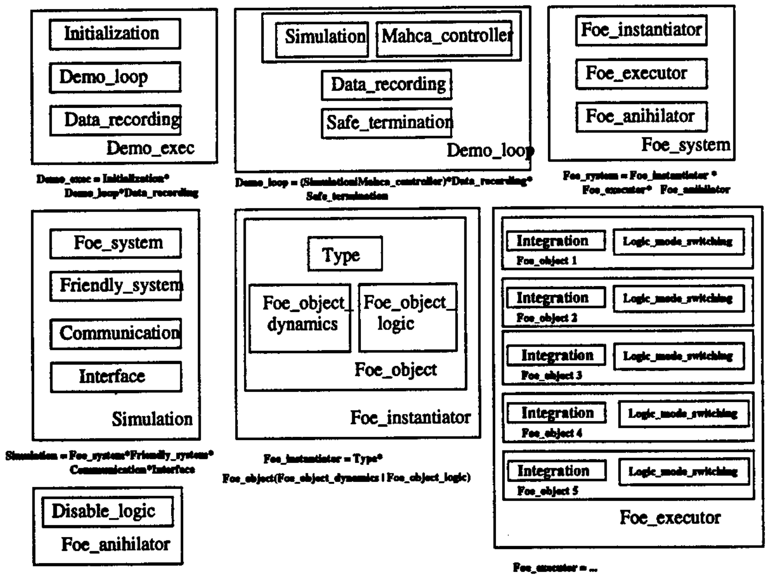


Figure 34: Sample Relational Terms for the Target Engagement Process

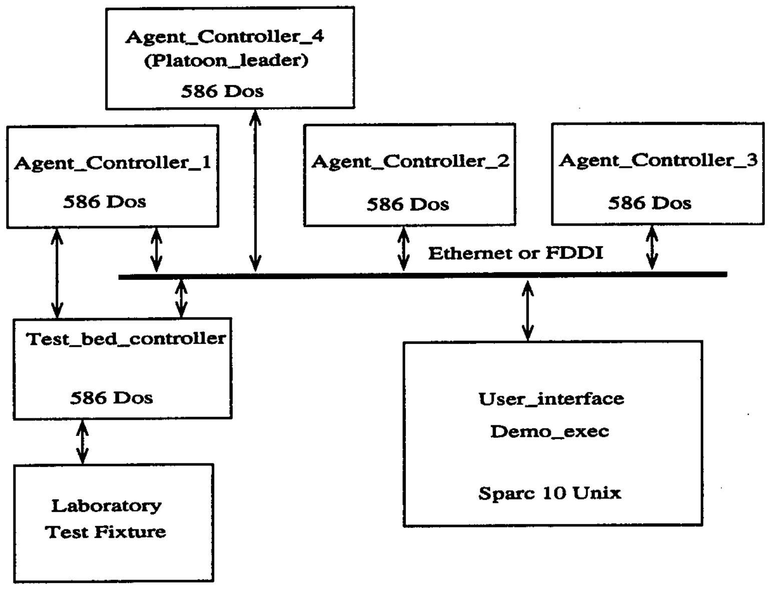


Figure 35: Hardware Requirements

The MAHCA agents were implemented in Quintus prolog. Drivers for the Agent and I/O Buses are implemented in WATCOM-C. While we initially intended that the dynamic behavior of one of the attack systems be emulated with the Laboratory Test Fixture (ATB-1000), in the actual implementation all of the agents will be simulated.

Some of the complexity of the target engagement process is contained in Figures 15 through 35 above. Our aim in implementing this model is not to capture the entire complexity of the problem of coordinating the engagement of multiple targets by multiple weapons systems. However, our aim is to model a significant portion of that complexity and also to create an extensible architecture and associated processes and technologies such that an engineering process is established for modeling, analysis and design of applications in the target engagement domain.

6 Conclusions

Our formulation gives a precise statement of the DIS communication network control problem in terms of a multiple agent hybrid control architecture. Our approach characterizes the problem via a knowledge base of equational rules that describe the dynamics, constraints and requirements of the DIS communications network (channels, switching modes, customer characteristics, scheduling and planning strategies, etc.).

We have developed a canonical representation of interacting networks of controllers. Given a connectivity graph with N nodes (controllers) and the corresponding agent's knowledge bases, a network of 2N agents can be constructed with the same input-output characteristics, so that each agent interacts only with another (equivalent) companion agent, whose knowledge base is an abstraction of the knowledge in the network. Thus, in general, the multiple-agent controller for any network configuration is reduced to a set of agent pairs.

One agent of the agent pair maintains coordination with other agent pairs across the network. We call that agent of the pair which represents network information the Theveninn Agent, after the author of a similar theorem in electrical network theory. The proof carried out by the Theveninn Agent generates, as a side effect, coordination rules that define what and how often to communicate with other agents. These rules also define what the controller needs from the network to maintain intelligent control of its physical plant.

Our approach develops a canonical way to prove the theorem characterizing the desired behavior for each agent by constructing and executing on-line a finite state machine called the "proof automaton." The inference process is represented as a recursive variational problem in which the criterion is an integral of a function called the Generalized Lagrangian or Demand functions. The Generalized Lagrangian maps the Cartesian product of equational rules and inference principles to the real line, thus effectively providing a hill-climbing heuristic for the inference strategy of the theorem prover (see section 3). In MAHCA the inference steps play a role analogous to action signals in conventional control. while vector fields on the manifold M constitute generators of feedback laws.

References

[1] Alur, R., Henzinger, T.A., and Sontag, E.D., eds., *Hybrid Systems III*, Lecture Notes in Computer Science vol. 1066, Springer-Verlag, (1996).

[2] Antsaklis, P., Kohn, W., Nerode, A., and Sastry, S. eds., *Hybrid Systems II*, Lecture Notes in Computer Science vol. 999, Springer-Verlag, (1995).

[3] Antsaklis, P., Kohn, W., Nerode, A., and Sastry, S. eds., *Hybrid Systems IV*, Lecture Notes in Computer Science vol. 1273, Springer-Verlag, (1997).

[4] Antsaklis, P.J., Stiver, J.A., and Lemmon, M.D., "Hybrid System Modeling and autonomous control systems," in [18], (1993), 366-392.

[5] Benveniste, A and Äström. K.M., "Meeting the Challenge of Computer Science in the Industrial Application of Control: An Introductory Discussion to the Special Issue," IEEE Transactions on Automatic Control, Vol. 38, pp. 1004-1010, 1993.

[6] Bott, R and Tu, L. W. *Differential Forms in Algebraic Topology*, Springer Verlag, New York, (1982).

[7] Branicky, M.S., " General hybrid dynamical systems: modleling, analysis, and control," in [1], (1996), 186-201.

[8] Branicky, M.S., Borkar, V.S., and Mitter, S.K., "A unified framework for hybrid control," In Proc. IEEE Conf. Decision and Control, Lake Buena Vista, FL, Dec 1994, 4228-4234.

[9] Crossley, J.N., Remmel, J.B., Shore, R.A. and Sweedler, M.E., *Logical Methods* Birkhauser, (1993).

[10] Deshpande, A., Godbole, D., Göllü, and Varaiya, P., "Design and evaluation tools for automated highway systems," in [1], (1996), 138-148.

[11] Deshpande, A. and Varaiya, P., "Viable control of hybrid systems," in [2], (1995), 128-147.

[12] Eilenberg, S., Automaton, Laguages, and Machines, Volume A, Academic Press, New York and London, (1974).

[13] Garcia, H.E. and A. Ray "Nonlinear Reinforcement Schemes for Learning Automata" Proceedings of the 29th IEEE CDC Conference, Vol. 4, pp 2204- 2207, Honolulu, HA, Dec. 5-7, 1990.

[14] Ge, X., Kohn, W., Nerode, A. and Remmel, J.B., "Algorithms for Chattering Approximations to Relaxed Optimal Control," MSI Tech. Report 95-1, Cornell University. (1995)

[15] Ge, X., Kohn, W., Nerode, A. and Remmel, J.B.,"Hybrid Systems: Chattering Approximations to Relaxed Control," *Hybrid Systems III*, (R. Alur, T.A. Henzinger, E.D. Sontag, eds.) Lecture Notes in Computer Science **1066**, Springer, (1996), 76-100.

[16] Gelfand, I.M. and Fomin, S.V., *Calculus of Variations*, Prentice Hall, 1963.

[17] Grossman, R.L., Nerode, A., and Kohn, W., "Nonlinear Systems, Automata, and Agents: Managing their Symbolic Data Using Light Weight Persistent Object Managers," Proceedings of FGCS'94, Workshop on Heterogeneous Cooperative Knowledge Bases, Kaxumans Yokata ed., Springer Verlag (1994).

[18] Grossman, R.L., Nerode, A., Ravn, A. and Rischel, H. eds., *Hybrid Systems*, Lecture Notes in Computer Science 736, Springer-Verlag, (1993).

[19] Grossman, R.L., Valsamis, D., and Qin, X., Persistent Object Stores and hybrid systems," Proceedings of the 32nd IEEE Conference on Decision and Control, IEEE Press, (1993), 2298-2302.

[20] Jacobson, N. *Lie Algebras*, Interscience NY. (1962).

[21] Kohn, W., "A Declarative Theory for Rational Controllers" Proceedings of the 27th IEEE CDC, Vol. 1, pp 131-136, Dec. 7-9, 1988, Austin, TX.

[22] Kohn, W., "Application of Declarative Hierarchical Methodology for the Flight Telerobotic Servicer" Boeing Document G-6630-061, Final Report of NASA- Ames research service request 2072, Job Order T1988, Jan. 15, 1988.

[23] Kohn, W., "Rational Algebras; a Constructive Approach" IR&D BE-499, Technical Document D-905-10107-2, July 7, 1989.

[24] Kohn, W., "The Rational Tree Machine: Technical Description & Mathematical Foundations" IR&D BE-499, Technical Document D-905-10107-1, July 7, 1989.

[25] Kohn, W., "Declarative Hierarchical Controllers", Proceedings of the Workshop on Software Tools for Distributed Intelligent Control Systems, pp 141-163, Pacifica, CA, July 17-19, 1990.

[26] Kohn, W., "Declarative Multiplexed Rational Controllers" Proceedings of the 5th IEEE International Symposium on Intelligent Control, pp 794-803, Philadelphia, PA, Sept. 5, 1990.

[27] Kohn, W., "Declarative Control Architecture" CACM Aug 1991,Vol34, No8.

[28] Kohn, W., "Advanced Architectures and Methods for Knowledge-Based Planning and Declarative Control" IR&D BCS-021, ISMIS'91, Oct. 1991.

[29] Kohn, W. and Murphy, A., "Multiple Agent Reactive Shop Floor Control" ISMIS'91, Oct. 1991.

[30] Kohn W., "Multiple Agent Inference in Equational Domains Via Infinitesimal Operators" Proc. Application Specific Symbolic Techniques in High Performance Computing Environment". The Fields Institute, Oct 17-20 1993.

[31] Kohn W., "Multiple Agent Hybrid Control" Proc of the NASA-ARO Workshop on formal Models for Intelligent Control, MIT,sept 30- Oct2, 1193.

[32] Kohn W., James J. and Nerode A., "Multiple-Agent Hybrid Control Architecture for Distributed Interactive Simulation" Proc.of ARO Workshop on Hybrid Sytems and Distributed Interactive Simulations. pp 100-140, Feb 28, March 1, 1994 Research Triangle Park, N.C.

[33] Kohn, W., James, J., Nerode, A., Harbison, K., and Agrawala, A., "A Hybrid Systems Approach to Computer-Aided Control Engineering", IEEE Control Systems, (1995), 14-24.

[34] Kohn, W., James, J., Nerode, A., and Lu, J., "Multiple-Agent Hybrid Control Architecture for the Target Engagement Process", Intermetric Technical Report, 1994.

[35] Kohn, W. and Nerode, A., "Multiple Agent Declarative Control Architecture" Proc. of the workshop on Hybrid Systems, Lygby, Denmark, Oct 19-21, 1992.

[36] Kohn, W. and Nerode, A., "Multiple Agent Hybrid Control Architecture" In [18], (1993), 297-316.

[37] Kohn W., and Nerode, A., "Multiple-Agent Hybrid Systems" Proc. IEEE CDC 1992, vol 4, pp 2956, 2972.

[38] Kohn, W. and Nerode, A., "An Autonomous Systems Control Theory: An Overview" Proc. IEEE CACSD'92, March 17-19, Napa, Ca.,pp 200- 220.

[39] Kohn W,, and Nerode A. "Models For Hybrid Systems: Automata, Topologies, Controllability, Observability," in [18], (1993) 317-356.

[40] Kohn W. and Nerode, A., "Multiple Agent Autonomous Control-A Hybrid Systems Architecture," In [9], (1993) 593-623.

[41] Kohn, W., Nerode, A. and Remmel, J.B., " Hybrid Systems as Finsler Manifolds: Finite State Control as Approximation to Connections," in [2], (1995), 294-321.

[42] Kohn, W., Nerode, A. and Remmel, J.B., "Feedback Derivations: Near Optimal Controls for Hybrid Systems," Proceedings of CESA'96 IMACS Multiconference, Vol 2. 517-521.

[43] Kohn, W., Nerode, A. and Remmel, J.B., "Continualization: A Hybrid Systems Control Technique for Computing," Proceedings of CESA'96 IMACS Multiconference, Vol 2. 507-511.

[44] Kohn, W., Nerode, A., Remmel, J.B., and Ge, X., "Multiple agent hybrid control: carrier manifolds and chattering approximations to optimal control," Proceedings of the 33rd IEEE Conference on Decision and Control (1994), 4221-4227.

[45] Kohn, W. and Remmel, J.B., "Digital to Hybrid Program Transformations", Proceedings of the 1996 IEEE International Symposium on Intelligent Control, 342-347.

[46] Kohn, W., and Remmel, J.B., "Implementing Sensor Fusion Using a Cost Based Approach,", to appear in the Proceedings of ACC'97.

[47] Kohn, W., Remmel, J.B., and Nerode, A.," Scalable Data and Sensor Fusion via Multiple-Agent Hybrid Systems," submitted to IEEE Transactions on Automatic Control.

[48] Kohn, W., Remmel, J.B. Brayman, V., James, J., and Nerode, A., "Advanced Nonlinear and Hybrid Systems Control Technology, Sagent Technical Progress Report, U.S. Army Armaments Research Development and Engineering Center, Contract Number DFAAE30-96-C-0042, August 1996.

[49] Kohn, W., Remmel, J.B., Moser, W.R., and Cummings, B., "Free flight ATC using hybrid agent systems," to appear in the Proceeding of 1997 Conference on Decision and Control, San Deigo, CA.

[50] Kohn, W. and T. Skillman, "Hierarchical Control Systems for Autonomous Space Robots," Proceedings of AIAA Conference in Guidance, Navigation and Control, Vol. 1, pp 382-390, Minneapolis, MN, Aug. 15-18, 1988.

[51] Kowalski, R., *Logic for Problem Solving*, North Holland, NY, 1979.

[52] Kuich, W. and Salomaa, A., *Semirings, Automata, Languages*, Springer Verlag, NY., 1985.

[53] Lygeros, J., Godbole, D.N., and Sastry, S., "A game-theoretic approach to hybrid system design," in [1], (1996), 1-12.

[54] Lygeros, J., Tomlin, C., and Sastry, S., "Multiobjective hybrid controller synthesis," in *Hybrid and Real Time Systems* (O. Maler, ed.), International Workshop Hart'97, Grenoble, France, March 26-28, 1997 Proceedings, Lecture Notes in Computer Science 1201, (1997), 109-123.

[55] Lloyd, J.W. *Foundations of Logic Programming*, second extended edition, Springer Verlag, NY, 1987.

[56] Liu, J.W.S., "Real-Time Responsiveness in Distributed Operating Systems and Databases" proceedings of the Workshop on Software Tools for Distributed Intelligent Control Systems, Pacifica, CA., July 17-19, 1990, pp 185-192.

[57] Nii, P.H., "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures" the AI Magazine, Vol. 7, No. 2, Summer 1986, pp 38-53.

[58] Neustad L. *Optimization: A theory of Necessary Conditions*, Princenton, U. Press, (1976).

[59] Padawitz, P., *Computing in Horn Clause Theories*, Springer Verlag, NY, 1988.

[60] Robinson, J.A., *Logic: Form and Function*, North Holland, NY, 1979.

[61] Skillman, T. and Kohn, W., et.al., "Class of Hierarchical Controllers and their Blackboard Implementations" Journal of Guidance Control & Dynamics, Vol. 13, N1, pp 176-182, Jan.-Feb., 1990.

[62] Tomlin, C., Pappas, G., Lygeros, J., Godbole, D., and Sastry, S., "Hybrid control models of the next generation air traffic control management," in [3], (1997), 378-405

[63] Warner, F.W., *Foundations of Differential Manifolds and Lie Groups*, Scott-Foresman, Glenview, Ill.

[64] Warga, K., *Optimal Control of Differential and Functional Equations*, Academic Press, NY., 1977.

[65] Young, L.C., *Optimal Control Theory*, Chelsea Publishing Co., NY, 1980.

[66] DIS Steering Committee, The DIS Vision A Map to the Future of Distributed Simulation (comment draft), (Margaret Loper, UCF, Chair; Steve Seidensticker, SAIC, DIS Vision Document Coordinator), Oct. 1993.

[67] NIU Forum Document Number BB-93-01, Interactive Simulation Applications Profile (Draft), 1993.

[68] Major General Wesley K. Clark "Digitization: Key to Landpower Dominance" Army, Vol. 43, No. 11, pp. 28-33.

[69] General Maxwell R. Thurman and Lt. Gen. William Hertzog "Simultaneity: The Panama Case" Army, Vol. 43, No. 11, pp. 16-24.

[70] General Jimmy D. Ross "Legacy for '90s in Louisiana Maneuvers" Army, Vol. 43, No. 6, pp. 16-20.